2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

An Empirical Performance Evaluation of GPU-Enabled Graph-Processing Systems

Yong Guo*, Ana Lucia Varbanescu§, Alexandru Iosup* and Dick Epema*
*TU Delft, The Netherlands, Email: {Yong.Guo, A.Iosup, D.H.J.Epema}@tudelft.nl
§University of Amsterdam, The Netherlands, Email: A.L.Varbanescu@uva.nl

***Abstract*—Graph processing is increasingly used in knowledge economies and in science, in advanced marketing, social networking, bioinformatics, etc. A number of graph-processing systems, including the GPU-enabled Medusa and Totem, have been developed recently. Understanding their performance is key to system selection, tuning, and improvement. Previous performance evaluation studies have been conducted for CPU-based graph-processing systems, such as Giraph and GraphX. Unlike them, the performance of GPU-enabled systems is still not thoroughly evaluated and compared. To address this gap, we propose an empirical method for evaluating GPU-enabled graph-processing systems, which includes new performance metrics and a selection of new datasets and algorithms. By selecting 9 diverse graphs and 3 typical graph-processing algorithms, we conduct a comparative performance study of 3 GPU-enabled systems, Medusa, Totem, and MapGraph. We present the first comprehensive evaluation of GPU-enabled systems with results giving insight into raw processing power, performance breakdown into core components, scalability, and the impact on performance of system-specific optimization techniques and of the GPU generation. We present and discuss many findings that would benefit users and developers interested in GPU acceleration for graph processing.**

I. Introduction

Many graph-processing algorithms have been designed to analyze graphs in industry and academic applications, for example, item and friend recommendation in social networks [1], cheater detection in online games [2], and subnetwork identification in bioinformatics [3]. To address various graphs and applications, many graph-processing systems have been developed on top of diverse computation and storage infrastructure. Among them, GPU-enabled systems promise to significantly accelerate graph-processing [4]. Understanding their performance, for example to select, tune, and extend these systems, is very challenging. Previous studies [5], [6] have investigated the performance of popular CPU-based distributed systems, such as Giraph [7], GraphLab [8], and Hadoop [9]. However, few of them include GPU-enabled systems. To address this problem, in this work we conduct the first comprehensive assessment of GPU-enabled graph-processing systems (including GPU-only and hybrid CPU and GPU systems).

We have identified three dimensions of diversity that complicate the performance evaluation of graph-processing systems in our previous work [5]. *Dataset diversity* originates from the variety of application areas for graph processing, from genomics to social networks, from citation databases to online games, all of which may create unique graph structures and characteristics. *Algorithm diversity* is the result of the different insights and knowledge that users and analysts want to gain from their graphs—a large number of algorithms have been developed for calculating basic graph metrics, for searching for important vertices, for detecting communities, etc. *System diversity* derives from the uncoordinated effort of different groups of developers who try to solve specific graph-processing problems, while tuning for their existing hardware infrastructure. Many graph-processing systems have appeared in recent years, from single-node systems such as GraphChi [10] and Totem [11] to distributed systems such as Giraph [7]; from generic systems such as Hadoop [9] to graph specific systems such as GraphX [12]; from CPU-based systems such as GraphLab [8] to *single-node* GPU-enabled systems such as Medusa [13] and MapGraph [14]. (To date, no mature, distributed graph-processing system using GPUs is publicly available.)

Understanding the performance of single-node GPU-enabled systems is important for two main reasons. Firstly, we argue that many datasets already fit to be processed in-memory on such systems. This corresponds, for example, to the datasets in use at many Small and Medium Enterprise (SMEs), and thus may affect up to 60% of the entire industry revenue [15]. Secondly, single-node systems are representative for, performance-wise, and the basic building block of future GPU-clusters for graph processing.

Understanding the performance of graph-processing systems is difficult. There is no analytical approach to understand their performance comprehensively. Thus, experimental performance evaluation studies [5], [6] have been recently proposed. However, they do not cover GPU-enabled systems. Moreover, many new challenges, such as different formats of in-memory graph representations, many optional optimization techniques provided by GPU-enabled systems, and different types of equipped GPUs, make it challenging to thoroughly understand the performance of GPU-enabled systems.

Our vision [16] is a four-stage empirical method for the performance evaluation of any graph-processing system. In this work, we extend our previous method [5] for evaluating graph-processing systems to include GPU-enabled systems. We define several new performance metrics to comprehensively evaluate the interesting performance aspects of GPU-enabled graph-processing systems—raw processing power, performance breakdown, scalability, the impact on performance of system-specific optimization techniques and of the

978-1-4799-8006-2/15 $31.00 © 2015 IEEE
DOI 10.1109/CCGrid.2015.20

GPU generation. We then conduct experiments, by implementing 3 typical graph algorithms and selecting 9 datasets with different structures, on 3 GPU-enabled systems—Medusa, Totem, and MapGraph. Our main contributions are:

1) We propose a method for the performance evaluation of GPU-enabled graph-processing systems (Section II). This method extends significantly our previous work for evaluating the performance of graph-processing systems, by defining new performance aspects and metrics, and by selecting new datasets and algorithms.
2) We demonstrate how our method can be used for evaluating and comparing GPU-enabled systems in practice. We setup comprehensive experiments (Section III), which we then conduct for three GPU-enabled graph-processing systems (Section IV). Last, we also identify, for these systems, various highlights and limitations (Section V).

II. Extended Method for GPU-Enabled Graph-Processing Systems

Our previous method for the performance evaluation of graph-processing systems [5] consists of four main stages: identifying interesting and important performance aspects and metrics; defining and selecting workloads with representativeness and coverage; implementing, configuring, and executing the experiments; and analyzing and presenting results in standard format. To address the challenges of the performance evaluation of GPU-enabled systems, in this section we adapt and extend our previous method, by identifying new performance aspects and metrics, and by selecting and including new datasets and algorithms.

A. Performance Aspects, Metrics, Process

To understand the performance of GPU-enabled systems, we identify five important performance aspects: three of them are adapted from our previous work, and two are newly designed relative to our previous work. For each aspect, we use at least one performance metric to quantify and characterize system performance. We further adapt for GPU-enabled systems the process for measuring and calculating the metrics.

We consider five performance aspects in this work:

1) *Raw processing power* (adapted from previous work): reflects the user perception of how quickly graphs are processed. We report in this work the run time of the algorithm for GPU-enabled systems.
2) *Performance breakdown* (adapted from previous work): the algorithm run time is not sufficient to understand all the details of system performance. Breaking down the total execution time into separate processing stages (system initialization time, algorithm run time, and data transfer time) enables the in-depth comparison of systems and, possibly, the identification of bottlenecks.
3) *Scalability* (adapted from previous work): the ability of a system to maintain its performance behavior when resources are added to its infrastructure. In our method, we test the vertical scalability (by adding GPUs) of systems in both strong and weak scaling. The observed changes in performance depend on both the number of added GPUs, and the algorithm and dataset. Indirectly, scalability allows us to reason about how well do graph-processing systems utilize accelerators.
4) *The impact on performance of system-specific optimization techniques* (newly designed): systems that can use different types of computing resources, for example both CPUs and GPUs, allow programmers to provide different implementations of the same algorithm, optimized for the different hardware. We study the impact of such optimizations (e.g., load balancing, graph representation), to understand their impact on system performance.
5) *The impact on performance of the GPU generation* (newly designed): several GPU generations are currently present in computing infrastructures, from mobile devices to servers and clusters. Understanding the correlation between their characteristics (compute capability, the number of cores, and memory capacity) and the system performance could guide users towards optimal (cost, performance) choices for their applications.

We summarize in Table I the performance metrics used to quantify the five performance aspects. For each of the metrics, we define how it can be obtained: by direct measurement or by calculation using measured parameters and dataset (i.e., graphs) properties. We define the total execution time (T_E) as the time from submission until completion. For each submission, we do not write output data to disk, but transfer the output data from GPUs to host memory. Algorithm run time (T_A) is the time used for actually executing the graph algorithms. Total execution time can be divided into times for different processing stages of the whole execution, including graph and configuration initialization time (T_I), algorithm run time (T_A), data transfer time from device to host (T_{D2H}), and overhead time (T_O) which includes the overhead in the initialization stage and the clear up stage. T_I can be further split into (1) graph initialization time (T_{I-G}, which includes reading and building graph in the host memory and transferring the graph data from host to device (T_{H2D})) , and (2) algorithm configuration time (T_{I-C}, which includes setting up algorithm-related configuration parameters and initial values in GPUs). We formulate the relationship of the total execution time and its breakdown as follows:

$T_E = T_{I-G} + T_{I-C} + T_A + T_{D2H} + T_O$

We define Edges Per Second (EPS) as the ratio between the number of all edges of the executed algorithm and the algorithm run time. EPS is a straightforward extension of the TEPS metric used by Graph500 [17]. To investigate the performance per computing unit, we further define the performance metric Normalized Edges Per Second (NEPS) as the ratio between EPS and the total number of computing units (GPUs in this work). For the same algorithm running on the same dataset, but with different setups (optimization techniques and GPU generations), we define the speedup as the ratio between the algorithm run time of baseline (see Section IV-D and IV-E for our baseline settings) and that of a different setup.

TABLE I
SUMMARY OF PERFORMANCE METRICS USED IN THIS STUDY.

Metric	How measured?	Derived	Relevant aspect (use)
Total execution time (T_E)	Total time of the full execution	-	Raw processing power (Table VI)
Algorithm run time (T_A)	Time of the algorithm running	-	Raw processing power (Figure 1, 2, 3)
Time breakdown	Time of the detailed execution	-	Performance breakdown (Table VI)
Strong scaling	T_A of multiple GPUs (N) for the same graph	-	Scalability (Figure 4)
Weak scaling	T_A of multiple GPUs (N) for different graphs	-	Scalability (Figure 6)
Normalized edges per second (NEPS)	-	$\#E/T_A/N$	Scalability (Figure 5)
Speedup	-	T_A/T'_A	Optimization, GPUs (Figure 7, 8, 9)

$\#E$ is the number of all edges of the executed algorithm. T'_A is the algorithm run time of a different setup.

B. Selection of graphs and algorithms

In this section, we discuss our selection of graphs and algorithms, which we used to evaluate the GPU-enabled graph-processing systems.

1) Graph selection: We select nine different graphs and summarize their information in Table II. We select both directed (Amazon, WikiTalk, and Citation) and undirected graphs (KGS, DotaLeague, Scale-22 to Scale-25). To comply with the requirement of many GPU-enabled systems, we need to store input undirected graphs in a directed manner. Thus, for each undirected edge, we create two directed edges as an equivalent. The selected graphs match well to the datasets used by SMEs in terms of scale and diversity. The graphs are from diverse sources (e-business, social networks, synthetic graphs), and different characteristics (e.g., high vs. low average degree, directed and undirected graphs). The Scale-22 to Scale-25 are undirected graphs created by the Kronecker generator introduced in Graph500 [17], with the scale from 22 to 25 and edge factor of 16. The other graphs have been collected from real-world applications, and have been shared through the Stanford Network Analysis Project (SNAP) [18]) and the Game Trace Archive (GTA) [2].

2) Algorithm selection: Considering the simplicity of the programming model of several GPU-enabled systems, we avoid algorithms using complex messages and mutating graph structures. Based on our comprehensive survey of graph-processing algorithms and applications [19], we select BFS, PageRank, and WCC as representative for three popular algorithms classes: graph traversal (used in Graph500), general statistics, and connected components, respectively. We summarize the characteristics of these algorithms in Table III.

Breadth First Search (BFS) is a widely used algorithm in graph traversal. BFS can be used as a building block for more complex algorithms, such as all-pairs shortest path and item search. BFS is a textbook algorithm. The PageRank algorithm (PageRank) is originally designed to rank websites in search engines. It can also be used to compute the importance of vertices in a graph. Several versions of PageRank have been proposed. In this work, we use the version described in Medusa [13]. Weakly Connected Component (WCC) is an algorithm for extracting groups of vertices connected via graph edges. For directed graphs, we say a group of vertices is weakly connected if any vertex in this group can be linked by an edge (no matter the direction) to another vertex in this group. We select in this work an implementation of WCC created by Wu and Du [20].

TABLE II
SUMMARY OF DATASETS USED IN THIS STUDY.

Graphs	V	E	d	$\bar{D}$	Max D
Amazon (D)	262,111	1,234,877	1.8	5	5
WikiTalk (D)	2,388,953	5,018,445	0.1	2	100,022
Citation (D)	3,764,117	16,511,742	0.1	4	770
KGS (U)	293,290	22,390,820	26.0	76	18,969
DotaLeague (U)	61,171	101,740,632	2,719.0	1,663	17,004
Scale-22 (U)	2,394,536	128,304,030	2.2	54	163,499
Scale-23 (U)	4,611,439	258,672,163	1.2	56	257,910
Scale-24 (U)	8,870,942	520,760,132	0.7	59	406,417
Scale-25 (U)	17,062,472	1,047,207,019	0.4	61	639,144

V and **E** are the vertex count and edge count of the graphs. **d** is the link density ($\times 10^{-5}$). $\bar{\mathbf{D}}$ is the average vertex out-degree. **Max D** is the largest out-degree. (D) and (U) stands for the original directivity of the graph. For each original undirected graph, we transfer it to directed graph (see Section II-B1).

TABLE III
SUMMARY OF ALGORITHMS USED IN THIS STUDY.

Algorithm	Main features	Use
BFS	iterative, low processing	building block
PageRank	iterative, medium processing	decision-making
WCC	iterative, medium processing	building block

TABLE IV
SUMMARY OF SYSTEMS USED IN THIS STUDY.

System	Version	Type	Release date
Medusa	Medusa-0.2	Multiple GPUs	2013-02
Totem	Trunk version	Hybrid, multiple GPUs	2014-08
MapGraph	MapGraph 0.3.2	Single GPU	2014-04

III. EXPERIMENTAL SETUP

In this section, we make a selection of GPU-enabled graph-processing systems, discuss the implementation of the graph-processing algorithms on the selected systems, and set the configuration of the parameters for running the algorithms.

A. System selection

Compared with the number of CPU-based graph-processing systems, there are fewer single-node GPU-enabled systems, and no distributed GPU-enabled graph-processing systems available for the public. Thus, in this work, we select three of the most mature single-node GPU-enabled graph-processing systems: Medusa, Totem, and MapGraph. Table IV summarizes our selected systems. We introduce each system in the following.

Medusa [13] is a graph-processing framework designed to help programmers use the GPU computing power with writing only sequential code. To achieve this goal, Medusa provides a set of user-defined APIs to hide the GPU programming details. Medusa can support multiple GPUs. Medusa extends the Bulk Synchronous Parallel (BSP) model by applying a "Edge-Vertex-Message" (EMV) model for each superstep. The EMV model breaks down the vertex-centric workload into separate chunks; the key concepts related to a chunk

are vertices, edges, and messages. Compared with a vertex-centric programming model, the fine-grained EMV model can achieve better workload balance of threads [13]. Medusa supports four different formats to store graphs in-memory: the vertex-oriented format Compressed Sparse Rows (CSR, or AA used in [13]), the edge-oriented format (ESL), the hybrid format (HY) of CSR and ESL, and the column-major adjacency array (MEG, or CAA used in [13]). HY and MEG are designed to reduce the uncoalesced memory access on GPUs. A graph-aware message buffer scheme is designed by Medusa to achieve better performance of processing messages between vertices. To maintain this buffer, a message index needs to be stored for each edge. For multiple GPUs, Medusa provides a multi-hop replication scheme and overlapping of computation and communication to alleviate the pressure from data transfers between partitions.

Totem [11] is a graph-processing system that can leverage both the CPU and the GPU (hybrid) as computing units by assigning graph partitions to them. Totem can also support multiple GPUs (with or without the CPU). Totem uses a vertex-centric programming abstraction under the BSP model. Each superstep of the BSP model includes three phases: a computation phase in which each computing unit executes the algorithm kernel on its assigned partitions, a communication phase in which each computing unit exchanges messages, and a synchronization phase to deliver all messages. Totem strictly uses CSR to represent graphs in-memory. To alleviate the cost of communication between partitions, Totem uses user-provided aggregation to reduce the amount of messages and maintains two sets of buffers on each computing unit for overlapping communication and computation. Totem implements other optimizations to improve the performance, for example, partitioning graphs by the vertex degrees and placing higher-degree vertices on CPU.

MapGraph [14] is an open-source project to support high-performance graph processing. The latest version (see Table IV) of MapGraph can only support a single GPU. MapGraph uses a modified Gather-Apply-Scatter model [8] to present each superstep of graph-processing algorithms. In the Gather phase, vertices collect updated information from in-edges and/or out-edges. During the Apply phase, every active vertex in the current superstep updates its value. In the Scatter phase, vertices send out messages to their neighbours. For each superstep, MapGraph maintains an array called frontier which consists of active vertices, to reduce the computation. The frontier for the next superstep is created in the Scatter phase of the current superstep. MapGraph uses CSR and the Compressed Sparse Column format (CSC [21], which is a reverse topology index of CSR) to store graphs. MapGraph adapts two strategies, dynamic scheduling and two-phase decomposition, to balance the workload of different threads.

Notation: From hereon, we use `M` for Medusa, `T-H` for Totem in hybrid mode using both the CPU and the GPU, `T-G` for Totem using the GPU(s) as the only computing resource, and `MG` for MapGraph.

B. System and experiment configuration

Hardware: We perform our experiments on DAS4 [22], which is a cluster with many different types of machines to cater different computation requirements of researchers in the Netherlands. We use four types of machines from DAS4 to conduct our experiments. Type 1 includes an Nvidia GeForce GTX 480 GPU (1.5 GB onboard memory) and an Intel Xeon E5620 2.4 GHz CPU. Type 2 is equipped with 8 Nvidia GeForce GTX 580 GPU (3 GB onboard memory per GPU) and dual Intel Xeon X5650 2.66 GHz CPUs. Type 3 consists of an Nvidia GeForce GTX 580 GPU (3 GB onboard memory) and an Intel Xeon E5620 2.4 GHz CPU. Type 4 has an Nvidia Tesla K20m GPU (5 GB onboard memory) and an Intel Sandy Bridge E5-2620 2.0 GHz CPU. The machines are used for different experiments as shown in Table V.

Algorithms: Some of the algorithms belong to libraries distributed with the systems, but the programming details may be different. When this is the case, we select a unique implementation, as described in Section II-B2. For each algorithm, we set the parameters identically on all systems. For BFS, we use the same source vertex for each graph on all systems. For PageRank, we consider maximum iteration as the only termination condition and set maximum iteration to 10 times. WCC does not need any specific parameter configuration.

System tuning: Several configuration parameters could be tuned in each system. The tuning of parameters can change the performance of these systems. We explore the influence of several common techniques for system tuning in SectionIV-D. For the other experiments, we use the default settings of each system. For example, for the hybrid mode of Totem, we place on the CPU higher-degree vertices, that is, the vertices whose total degree is about one third of the number of edges of the whole graph.

Dataset considerations: Because the WCC algorithm considers that two vertices are connected when there is an edge between them, when running the WCC algorithm for *directed* graphs (Amazon, WikiTalk, and Citation), we create a reverse edge for each pair of vertices which are originally connected by a single directed edge. The new datasets are *AmazonWCC*, *WikiTalkWCC*, and *CitationWCC*, with the number of edges 1,799,584, 9,313,364, and 33,023,481, respectively.

Further configuration and settings: For all systems, the GPU compiler is Nvidia CUDA 5.5. We use CUDPP 2.1 [23] and Intel TBB 4.1 [24] as third-party libraries for Medusa and Totem, respectively. We repeat each experiment 10 times, and we report the arithmetic mean. We only show error bars in our scalability test, because in all the other experiments our results from 10 runs are very stable, with the largest variance under 5%.

IV. Experimental results

In this section we present our experimental results. Table V summarizes our experimental setups. The experiments we have performed are:

TABLE V
EXPERIMENTAL SETUP FOR EACH EXPERIMENT IN SECTION IV.

Section	Systems	Algorithms	Datasets	Metrics	GPU (Machine Type)	Graph Formats
Section IV-A	All	All	6	Algorithm run time	GTX 480 (Type 1)	CSR
Section IV-B	All	All	2	Total execution time and its breakdown	GTX 480 (Type 1)	CSR
Section IV-C	Medusa, Totem	PageRank	4	Strong and weak scaling, NEPS	GTX 580 (Type 2)	CSR
Section IV-D	All	PageRank	6	Speedup	GTX 480 (Type 1)	CSR, HY, MEG
Section IV-E	All	All	6	Speedup	GTX 480, GTX 580, K20m (Type 1, 3, 4)	CSR

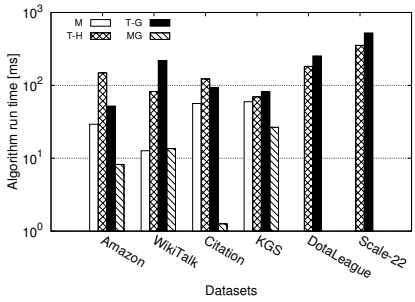

Fig. 1. The algorithm run time for BFS of 6 datasets on all systems. (Missing bars are explained in text.)

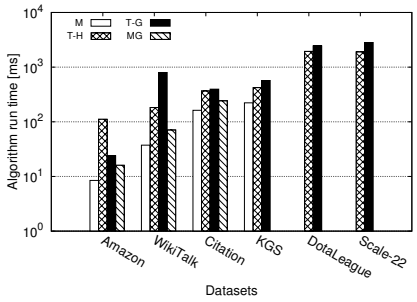

Fig. 2. The algorithm run time for PageRank of 6 datasets on all systems. (Missing bars are explained in text.)

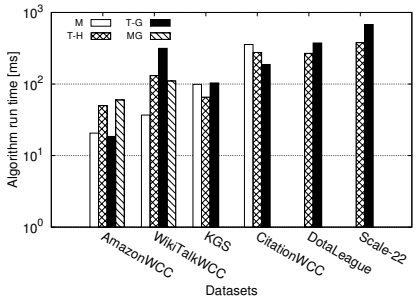

Fig. 3. The algorithm run time for WCC of 6 datasets on all systems.(Missing bars are explained in text.)

- Raw processing power (Section IV-A): we have measured the algorithm run time and we report it for all combination of algorithms, datasets, and systems.
- Performance breakdown (Section IV-B): we have analyzed the total execution time in detail. We show the breakdown of the total execution time as introduced in Section II.
- Scalability (Section IV-C): we have measured the vertical scalability of Totem and Medusa in both strong scaling and weak scaling.
- The impact on performance of system-specific optimization techniques (Section IV-D): we have applied system-specific optimization techniques and we report the impact they have on the performance of systems.
- The impact on performance of the GPU generation (Section IV-E): we have investigated the behavior of all three systems on three different generations of GPUs and we report the performance changes we have observed.

A. Raw processing power: algorithm run time

In this section, we reported a full set of experiments (all algorithms, all systems, and 6 datasets) and analyze the algorithm run time.

Key findings:

- Totem is the only system that can process all 6 datasets for all algorithms. Medusa and MapGraph crash for some of the setups.
- There is no overall best performer, but in most cases Totem performs the worst.
- The optimization techniques used by the graph-processing systems lead to inconsistent performance benefits across different algorithms.
- Relative to the performance we have observed on CPU-based systems [5], the results of the GPU-enabled systems studied in this work are significantly faster.

We report results for the NVIDIA GTX 480 GPU, which fits all real-world datasets and the Scale-22 synthetic one (experiments on larger datasets are discussed in Section IV-C). We show the algorithm run time, for each combination of setup parameters, in Figures 1, 2, and 3. For the horizontal axis of each figure, we order the datasets by their number of edges, from left to right.

We depict the performance of BFS in Figure 1. We see that only Totem can handle all 6 datasets. Medusa and MapGraph crash attempting to construct DotaLeague and Scale-22 in-memory and report an "out of memory" error. Although in these experiments, all systems use the CSR format to store graphs in-memory, the implementation details are different. Totem strictly represents the graph in the CSR format by using two arrays V and E: array V contains the start indices that can be used to fetch the neighbour lists, which are stored in array E. Medusa uses a structure of arrays, which includes extra data such as the number of edges for each vertex and the message index for each edge. In MapGraph, a graph is represented in both the CSR format and the CSC format in-memory. The usage of CSC doubles the memory consumption.

In Figure 1, the two modes of Totem have longer run times than either Medusa or MapGraph. An important reason for

this is the different parallelism granularities of the kernel. In Totem, the number of threads is the same as the number of vertices and each thread processes one vertex [4]. This mapping can result in workload imbalance because every thread's workload is skewed by the degree of its assigned vertex. The performance of Totem worsens when the vertex degrees are highly skewed, as seen for example, for running T-G on the WikiTalk dataset (the degrees of most vertices in WikiTalk are smaller than 10K, but there is one vertex with more than 100K neighbours). This imbalance can be alleviated by assigning these higher-degree vertices to the CPU, as shown by the result of T-H on Wikitalk. In Medusa, the whole workload is divided into three phases which target individual vertices, edges, and messages. Medusa uses a fixed amount of blocks and threads per block to process all vertices and all edges. The workloads of threads are well balanced by assigning vertices and edges to threads in turn. MapGraph adapts complex dynamic scheduling and two-phase decomposition to balance the workload of threads.

For the BFS algorithm (Figure 1), MapGraph performs best for all the graphs it can handle. We attribute this advantage to the design of a *frontier* which maintains active vertices for each superstep. For the BFS algorithm, the active vertices in each superstep can be significantly less than the full set of vertices. Thus, for each iteration of BFS on MapGraph, only a part of vertices are accessed and computed. Although Medusa and Totem do not compute non-active vertices, both systems have to access all vertices. The impact of the frontier is significant when the set of active vertices is small. For example, the algorithm run time of Citation (BFS traverse coverage is 0.1%) is much shorter than other datasets (BFS traverse coverages are greater than 98.5%). However, the implementation of a frontier may have negligible impact for other algorithms, and may even cause crashes of the system due to lack of memory.

For PageRank (see Figure 2), MapGraph cannot outperform Medusa with any dataset because all vertices are active in each superstep (according to our implementation of this algorithm, see Section II-B2). This is in contrast to our findings for BFS (Figure 1) regarding the impact of the frontier. Furthermore, MapGraph cannot process KGS because it doest not have enough memory for maintaining such a larger frontier.

The comparison of the algorithm run time of WCC on three systems is shown in Figure 3. Unlike the results of BFS and PageRank, the performance of Totem is not always worse than Medusa and MapGraph; Totem exhibits better performance in both hybrid mode and GPU-only mode on AmazonWCC, KGS, and CitationWCC. There are three main reasons that lead to this result. Firstly, a large amount of updated information needs to be send between supersteps of WCC. Totem aggregates the updated information sent to the same vertex that can reduce the inter-superstep communication time. Secondly, the distribution of vertex degrees of AmazonWCC, KGS, and CitationWCC is not highly skewed. Thirdly, the computation of the algorithm is not intensive. The second and third reasons result in a relatively balanced workload for each thread.

TABLE VI
THE BREAKDOWN OF RUNNING THE BFS ALGORITHM ON THE AMAZON DATASET AND THE WCC ALGORITHM ON THE AMAZONWCC DATASET. (ALL TIME VALUES IN MILLISECONDS.)

	BFS on Amazon				**WCC on AmazonWCC**			
	M	MG	T-H	T-G	M	MG	T-H	T-G
T_{I-G}	1278.1	1064.9	339.1	316.6	1787.1	1519.0	477.8	452.4
T_{I-C}		0.7	0.5	0.3		0.2	1099.9	1062.0
T_{H2D}	8.1	3.3	1.1	1.8	10.3	4.3	1.5	2.3
T_A	29.3	8.2	148.0	51.9	20.6	59.2	49.9	18.5
T_{D2H}	1.5	0.6	0.6	0.4	1.5	0.4	0.7	0.8
T_O	723.7	18.2	3.7	46.5	739.5	18.0	4.1	46.4
T_E	2032.5	1092.7	491.9	415.7	2548.8	1596.9	1632.5	1580.2

From Figures 1, 2, and 3, we also find that the structure of graphs have consistent impact on the algorithm run time of all three algorithms in two modes of Totem. For instance, the algorithm run time of T-H is always more than that of T-G on Amazon, while the algorithm run time of T-H is always less on WikiTalk, no matter which algorithm runs.

B. Performance breakdown

In this section, we report, for each algorithm, the total execution time and its breakdown.

Key findings:

- The time for reading the graph and for constructing the graph in-memory dominates the total execution time.
- The initialization time of the systems should be reduced.

Because the performance breakdown of PageRank is very similar to that of BFS, we show the breakdown of the total execution time on Amazon for just BFS and WCC in Table VI. Note that the input file of WCC is AmazonWCC, making the graph initialization time T_{I-G} for all systems longer than that of BFS on Amazon.

Overall, we notice that for all algorithms, the initialization time (including T_{I-G} and T_{I-C}) is the major part of the total execution. Thus, the performance of initialization is essential for improving the overall performance of such systems. Compared with Totem and MapGraph, Medusa needs more time for initialization because it reads the input file by words, not by lines, and even if a graph is not partitioned, the data structures for building partitions are created and calculated. Medusa also shows longer overhead, mainly caused by the initialization of system, such as configuring the L1 cache and shared memory of the GPU. The graph initialization and algorithm configuration in Medusa are aggregated. MapGraph uses a two-step procedure to build CSR and CSC formats in-memory. A graph input file is read into the Coordinate (COO) format [21], the tuples in COO are then sorted and the CSR and CSC formats are constructed from COO. However, when algorithms do not need the CSC format for execution, the time for building the CSC format is wasted.

For the WCC algorithm, each vertex is assigned an initial value using its vertex ID. However, in the hybrid mode of Totem, input graphs are partitioned and all vertices are re-assigned to new IDs in each partitions. Each partition keeps a map for mapping new IDs in this partition to original IDs in the input graph. Thus, for the configuration of WCC, Totem needs

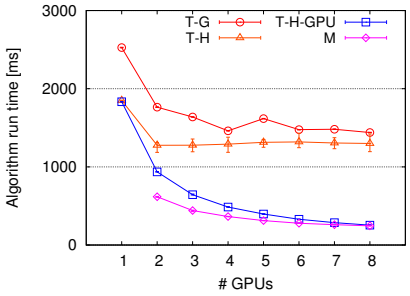

Fig. 4. The strong scaling of Totem and Medusa on Scale-22. (The missing point for Medusa, # GPUs =1, is explained in Section IV-A.)

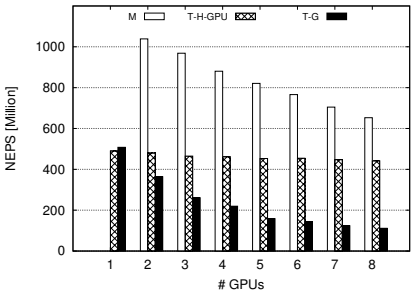

Fig. 5. The normalized edges per second of processing Scale-22. (The missing bar for Medusa, # GPUs =1, is explained in Section IV-A.)

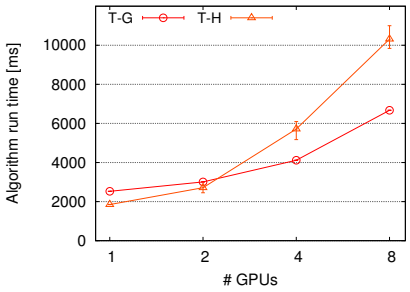

Fig. 6. The weak scaling of Totem.

to access the map once for initializing the value of each vertex. Totem does not support special mechanism for initializing one partition graph, thus the configuration time of the GPU-only mode is as high as that of the hybrid mode. Medusa faces the same long configuration problem when it uses multiple GPUs.

C. Evaluation of scalability

In this section, we evaluate the scalability of Totem and Medusa with using multiple GPUs.

Key findings:

- Totem and Medusa show reasonable strong and weak scaling with the increase of the number of GPUs.
- Increasing the number of GPUs may not always lead to performance improvement.

All experiments in this section are executed on a machine with 8 Nvidia GTX 580 3GB GPUs. We assign vertices randomly to GPUs. Our scalability tests are using PageRank because it is the most compute-intensive algorithm in this work as shown in Section IV-A. We test both strong scaling and weak scaling. For strong scaling, we use the Scale-22 graph which is the largest graph that can be handled by Totem with only one GPU. We increase the number of GPUs from 1 to 8. The strong scaling results for T-H include both the CPU workload (constant) and the GPUs workload (scaled). For weak scaling, we use four generated graphs from Scale-22 to Scale-25, and test them on 1, 2, 4, and 8 GPUs.

For strong scaling, we observe that Medusa exhibits better scalability than Totem (Figure 4). The algorithm run time of Medusa keeps decreasing by adding more GPUs. For Totem, in both T-H and T-G, the algorithm run time does not change too much after a certain number of GPUs. However, due to the random placement of vertices, the workload per GPU may not be balanced. Combined with the increasing time for communication, the decreasing trend of the algorithm run time is not obvious with adding GPUs. As we discussed in Section IV-A, the workload of each thread on a GPU is not well balanced for Totem, which may cause the bad scaling after using 4 GPUs in T-G. As for T-H, because the algorithm run time is dominated by the CPU computation, when using 2 GPUs or more, the performance remains almost constant. This can be proved by the longest algorithm run time of all GPUs (represent as T-H-GPU). We also notice that T-H shows more unstable behavior (note the error bars), which indicates that the algorithm execution on the GPUs is more stable than on the CPU.

We show in Figure 5 the normalized edges per second (NEPS) for Medusa and Totem. To compute NEPS, we first calculate EPS by dividing the algorithm run time by the number of edges accessed by PageRank, then we normalize EPS by the number of GPUs. For the hybrid mode of Totem, we only consider the scaled workload on GPUs, see T-H-GPU in Figure 5. We compute the EPS of T-H-GPU using the longest algorithm run time of all GPUs and the number of edges placed on GPUs (according to the configuration in Section III, about two thirds of edges are processed on GPUs). The NEPS of T-H-GPU remains relatively constant at around 450 million. Because the higher-degree vertices are mainly assigned to the CPU, the degrees of the vertices placed on the GPUs are not vary, leading to a relatively balanced workload for each GPU and for each thread on a same GPU.

The weak scaling of T-G is presented in Figure 6. We do not have results for Medusa because the experiments using Scale-22 to Scale-25 crashed due to the high memory consumption. T-H is only shown as a reference to T-G: due to the use of both a single CPU (workload is not scaled) and multiple GPUs (workload is scaled). The efficiency of weak scaling of T-G is 84%, 61%, and 38%, for using 2, 4, and 8 GPUs, respectively. The efficiency decreases because the total workload does not grow linearly with the increase of the graph scale, and this may also because the vertex degrees of larger graphs are more skewed (Table II).

D. Evaluation of system-specific optimization techniques

Many optimization techniques can be used to change the performance of the graph-processing systems studied in this work. In this section, we evaluate the performance of Medusa

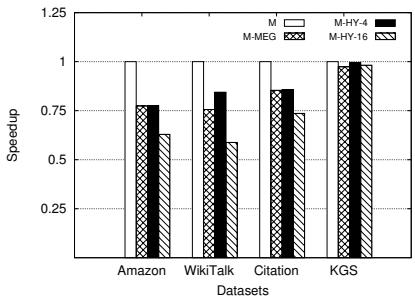

Fig. 7. The speedup of different graph representations on Medusa.

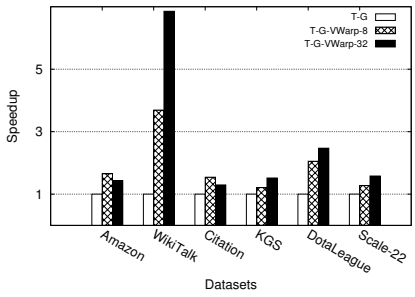

Fig. 8. The speedup of using different virtual warp-centric setups on Totem.

when storing the graph in-memory using different representations, the performance of Totem when using the different configurations for the virtual warp-centric technique [25], and the performance of MapGraph when using different thresholds for choosing its scheduling strategies.

Key findings:

- Performance improvements depend significantly on system-specific optimization techniques.
- Advanced techniques may not always have a positive impact of the performance on systems.

We choose PageRank because it is the most compute-intensive algorithm in our study. To focus on the performance of the GPU, we do not report result from T-H.

Medusa can support several graph representations, CSR, ESL, hybrid (HY) format of CSR and ESL, and MEG. HY requires a threshold value to calculate the proportion of CSR and ESL. Figure 7 shows the speedup of building graphs in the MEG format (M-MEG) and the HY format (with the threshold of 4 and of 16, represented as M-HY-4 and M-HY-16, respectively). The threshold values for M-HY are the default value (16) and representative for the average vertex degree (4). We set the performance of Medusa using the CSR format as the baseline. It is very surprising that all MEG and HY experiments have worse performance than the CSR format. This result is different from the result of [13] with running PageRank on a RMAT dataset on Medusa, in which MEG and HY are better. We cannot directly compare our results because as we shown in our previous work [5], results can be very sensitive to datasets. M-HY-4 significantly outperforms M-HY-16 on Amazon, WikiTalk, and Citation because the threshold is closer to their average vertex degrees, which is similar to the result of [13].

TABLE VII
SUMMARY OF THE GPU GENERATIONS USED IN THIS STUDY. SP/DP DENOTE SINGLE/DOUBLE PRECISION OPERATIONS.

	GTX 480	**GTX 580**	**K20m**
Frequency (GHz)	1.40	1.57	0.71
# Cores	480	512	2496
Peak GFLOPS (SP/DP)	1344.0/672.0	1603.6/801.8	3519.3/1173.1
Memory capacity (GB)	1.5	3	5
Peak Memory Bandwidth (GB/s)	177.4	193.0	208.0

To improve the performance of Totem, we use the virtual warp-centric technique to balance the workload of threads in the algorithm kernel. Figure 8 illustrates the speedup of the algorithm run time by using virtual warp (with virtual warp size of 8 and of 32, represented as T-G-VWarp-8 and T-G-VWarp-32, respectively). The warp size values are the default value (32) and representative for the graph properties (8). We set the performance of T-G as the baseline. From Figure 8, we notice that both T-G-VWarp-8 and T-G-VWarp-32 can obtain significant improvement, with the highest speedup of around 7. T-G-VWarp-8 has better performance than T-G-VWarp-32 on Amazon and Citation, whose average degrees are closer to 8. This finding matches to the result of the work of [25] with running BFS on several datasets.

We have also investigated the performance of tuning MapGraph. In the scatter phase, dynamic scheduling and two-phase decomposition can be used to create the frontier of the next superstep. MapGraph uses a threshold on the frontier size of the current superstep to determine which strategy would be executed. We tune the threshold for running PageRank on all datasets with various values (from 1 to 20000, default value 1000). The algorithm run time is not sensitive to the threshold, with a variance within 2% for different thresholds on the same dataset and algorithm. We further check the threshold influence on BFS and WCC, and we get the same result as PageRank.

E. Evaluation of the GPU generation

In this section, we run all experiments of Section IV-A on two other GPUs: GeForce GTX 580 and Tesla K20m.

Key findings:

- Memory consumption is a key issue of Medusa and MapGraph, when processing the largest graphs in our study.
- Using a GPU with improved processing capability can help, but not always.

The details of GPUs are introduced in Table VII. We compare the performance of systems deployed on different generations of GPUs. We present a representative selection of results from the whole set of experiments. To focus on the performance impact of the GPU generation, we do not report results from the hybrid mode of Totem in this section.

Table VIII shows, for WCC, the change of processable datasets of Medusa and MapGraph on GTX 480, GTX 580, and K20m. We choose the WCC algorithm because on GTX 480, it has the most number of datasets that cannot be processed on Medusa and MapGraph. We use "Y" to depict a dataset that can be processed and "N" for crashes. For each crash, we present the reason why it happens. "F" represents

TABLE VIII
THE SUCCESS OF RUNNING THE WCC ALGORITHM ON MEDUSA (M) AND MAPGRAPH (MG). ("Y" DENOTES SUCCESSFUL PROCESSING. ALL OTHER VALUES DENOTE CRASHES, SEE TEXT FOR DETAILS.)

		GTX 480	GTX 580	K20m
KGS	M	Y	Y	Y
	MG	N (R: M)	Y	Y
CitationWCC	M	Y	Y	Y
	MG	N (R: M)	N (R: M)	Y
DotaLeague	M	N (F)	N (R: E)	N (R: E)
	MG	N (F)	N (R: M)	N (R: M)
Scale-22	M	N (F)	N (F)	N(R: E)
	MG	N (F)	N (F)	N (R: M)

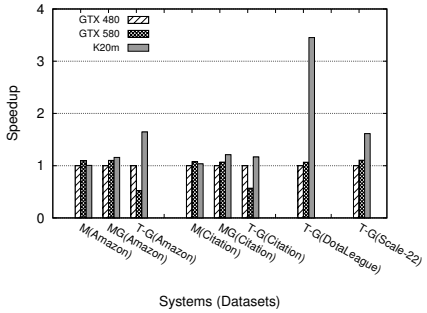

Fig. 9. The speedup of running PageRank using different GPU generations.

a crash that occurs when the graph cannot **F**it into the GPU memory at the initialization time. "R" represents a crash that happens during the **R**un time of WCC. For each R, we further detail if it is caused by out of **M**emory or CUDA **E**rror. From Table VIII, we find that more datasets can be processed by using GPUs with larger memory. For example, MapGraph can handle KGS and CitationWCC on K20m. We also observe that DotaLeague and Scale-22 still cannot be processed by Medusa and MapGraph. For DotaLeague and Scale-22 on MapGraph, all crashes are caused by out of memory, either in initializing the graph or running the algorithm with large frontier. For Medusa, although memory is not the bottleneck on K20m, CUDA errors are reported by its library CUDPP 2.1 [23].

In Figure 9, we report the obtained speedup for every system when using different GPU generations. We set the performance on GTX 480 as a baseline and we pick the PageRank algorithm because it has the longest algorithm run time in our study. For the datasets, we choose Amazon (smallest dataset), Citation (the largest dataset that can be processed by all systems on GTX 480), DotaLeague (the largest real-world dataset), and Scale-22 (synthetic graph). In most cases, the performance we observed increases when using more powerful GPUs. For Medusa and MapGraph, the performance improvement is not significant. We even find performance degradation: for instance, T-G's performance on GTX 580 is worse than that on GTX 480. Overall, we note that simply migrating systems to more powerful GPUs may not be sufficient to obtain much higher performance, as we would expect. To get better performance, additional parameter tuning may be performed.

V. USER EXPERIENCE

Performance is a key issue for selecting systems. From the perspective of an end user, the usability of the systems is also important. In this section, we discuss our user-experience with each of the selected system. We also compare the experience of using GPU-enabled systems with our prior experience as users of CPU-based systems.

For Medusa, as the designers of Medusa claim, we could use the fewest lines of code, relative to the other systems considered in this study, to implement the core part of the algorithms. However, it takes more lines of code and more effort to design the data structure for each of vertices, edges, and messages. The graph representation and the design of the message buffer are not memory-efficient, which limit the scale of the graphs that can be processed by Medusa. The errors reported by the third-party messaging library (see Section IV-E) reveal that Medusa needs further validation with this library. Learning how to use Medusa is not easy, because the documentation is scarce.

For Totem, we needed a majority of the lines of code for the core part of algorithms, because no high-level API is provided. We had to implement two different versions of every algorithm, for the CPU and for the GPU. The performance of Totem relies on the implementation of algorithms, as users have to address the details for coding the kernel for the GPU. Totem can process the largest datasets among the three systems, because it has efficient memory usage, and because it can use the storage resources of the host. The documentation about Totem is also scarce, but the clear structure of the project and a large number of algorithm examples can help users get familiar with the system better than Medusa.

MapGraph provides a set of APIs for users. Similarly to Medusa, users do not need to touch kernel-programming on the GPU. MapGraph lacks the ability to handle large datasets, as it is the most memory-consuming system in our study. MapGraph has better documentation than Medusa and Totem, but a comprehensive user tutorial is still needed. For the future, MapGraph promises to evolve towards a distributed system that can use GPU-clusters.

Compared with CPU-based systems [19], there are still many aspects to be improved in GPU-enabled graph-processing systems. We point out three main issues: the scale of graphs could be processed is rather small, primarily due to lack of memory; it is difficult to implement graph algorithms with complex message delivery and with graph structure mutation, because the GPU programming models may not be suitable for these aspects; the developer and user community is smaller and less active than for CPU-based systems.

VI. RELATED WORK

Motivated by the increasing practical need for graph-processing systems, many studies have focused on the performance evaluation of graph-processing systems in the past two years. Combined, this body of work compares the performance of many graph-processing systems, using many performance aspects and metrics, tens of diverse datasets, and various graph-processing algorithms and applications. However, individual studies rarely combine these desirable features of a performance evaluation study. This work complements all

these previous studies with a process that focuses on a new class of systems (GPU-enabled instead of CPU-based) and reports on a more diverse set of metrics.

Elser and Montresor [26] evaluate the performance of 5 distributed systems for graph processing but use only 1 graph algorithms and only 1 performance metrics. Our previous work [5] proposes an empirical method for benchmarking CPU-based graph-processing systems and reports the performance of the systems using more metrics and broader workload concerns. Han et al. [6] use similar performance metrics to [5] and focus on a family of Pregel-like systems. Lu et al. [27] include the influence of algorithmic optimizations and the characteristics of input graphs.

Most of the previous evaluation studies were proposed on CPU-based systems. Relatively few performance evaluation studies focus on GPU-enabled systems. Most of these studies were proposed by system designers to exhibit their system, and may lack the method or bias necessary for this kind of studies. Specifically, the previous studies on the performance of GPU-enabled systems lack representative workloads [28], performance metrics [13], [14], and comparative systems [11], [29]. Our study is the first in-depth performance evaluation study of GPU-enabled graph-processing systems.

VII. Conclusion

Using the capability of GPUs to process graph applications is a new promising branch of graph-processing research. A number of GPU-enabled graph-processing systems, with different programming models and various optimization strategies, have been developed, which raises the challenging question of understanding their performance. We conduct in this work the first comprehensive performance evaluation of GPU-enabled graph-processing systems.

This method significantly extends our previous work on the topic, by adapting performance aspects and introducing performance metrics that focus on GPU-enabled systems, by adding more datasets, and by focusing on important graph-processing algorithms. We focus on the following performance aspects: raw processing power, performance breakdown, scalability, the impact on performance of system-specific optimization techniques and of the GPU generation. We use at least one performance metric to quantify each performance aspect, such as total execution time and its breakdown to measure detailed performance, normalized metric NEPS to characterize scalability, etc. We select 9 datasets with diverse characteristics from both real-world domains and popular synthetic graph generators, up to scales of more than 1 billion directed edges. We also select 3 graph algorithms that are commonly used by the GPU graph-processing community.

We use the proposed method and report the first comprehensive performance evaluation and comparison of 3 GPU-enabled graph-processing systems, Medusa, Totem, and MapGraph. We show the strengths and weaknesses of each system and list key findings for each of our experiments. Overall, we conclude that understanding the performance of these systems can be very useful, but requires an in-depth experimental study.

Acknowledgment

This work is supported by the Dutch STW/NWO Veni personal grants @large (#11881) and Graphitti (#12480), by the Dutch national program COMMIT and its funded project COMMissioner, by the Dutch KIEM project KIESA, by the NSFC No.61379146, No.61272483, No.61272056, and by the Fund No.JC13-06-03. The authors wish to thank Hassan Chafi and the Oracle Research Labs for their generous support.

References

[1] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations," in *SIGKDD*, 2005.
[2] Y. Guo and A. Iosup, "The Game Trace Archive," in *NetGames*, 2012.
[3] T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel, "Discovering Regulatory and Signalling Circuits in Molecular Interaction Networks," *Bioinformatics*, 2002.
[4] P. Harish and P. Narayanan, "Accelerating Large Graph Algorithms on the GPU Using CUDA," in *HiPC*, 2007.
[5] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke, "How Well do Graph-Processing Platforms Perform? An Empirical Performance Evaluation and Analysis," in *IPDPS*, 2014.
[6] M. Han, K. Daudjee, K. Ammar, M. T. Ozsu, X. Wang, and T. Jin, "An Experimental Comparison of Pregel-Like Graph Processing Systems," *VLDB*, 2014.
[7] "Giraph." [Online]. Available: http://giraph.apache.org/
[8] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud," in *VLDB*, 2012.
[9] T. White, *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.
[10] A. Kyrola, G. Blelloch, and C. Guestrin, "GraphChi: Large-scale Graph Computation on Just a PC," in *OSDI*, 2012.
[11] A. Gharaibeh, E. Santos-Neto, L. B. Costa, and M. Ripeanu, "Efficient Large-Scale Graph Processing on Hybrid CPU and GPU Systems," *TOPC*, 2013.
[12] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph Processing in a Distributed Dataflow Framework," in *OSDI*, 2014.
[13] J. Zhong and B. He, "Medusa: Simplified Graph Processing on GPUs," *TPDS*, 2013.
[14] Z. Fu, M. Personick, and B. Thompson, "MapGraph: A High Level API for Fast Development of High Performance Graph Analytics on GPUs," in *GRADES*, 2014.
[15] "European Commission Annual Reports," in *ECORYS*, 2013.
[16] Y. Guo, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke, "Benchmarking Graph-Processing Platforms: a Vision," in *ICPE*, 2014.
[17] Graph500. [Online]. Available: http://www.graph500.org/
[18] "SNAP." [Online]. Available: http://snap.stanford.edu/index.html
[19] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke, "How Well do Graph-Processing Platforms Perform? An Empirical Performance Evaluation and Analysis: Extended Report," Delft University of Technology, Tech. Rep. PDS-2013-004, 2013.
[20] B. Wu and Y. Du, "Cloud-Based Connected Component Algorithm," in *ICAICI*, 2010, pp. 122–126.
[21] "MapGraph." [Online]. Available: http://mapgraph.io/
[22] "DAS4." [Online]. Available: http://www.cs.vu.nl/das4/
[23] "CUDPP." [Online]. Available: http://cudpp.github.io/
[24] "Intel Threading Building Blocks." [Online]. Available: https://www.threadingbuildingblocks.org/
[25] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun, "Accelerating CUDA Graph Algorithms at Maximum Warp," *SIGPLAN*, 2011.
[26] B. Elser and A. Montresor, "An Evaluation Study of Bigdata Frameworks for Graph Processing," in *Big Data*, 2013.
[27] Y. Lu, J. Cheng, D. Yan, and H. Wu, "Large-Scale Distributed Graph Computing Systems: An Experimental Evaluation," *VLDB*, 2014.
[28] Z. Fu, H. K. Dasari, B. Martin, and B. Thompson, "Parallel Breadth First Search on GPU Clusters," University of Utah, Tech. Rep., 2014.
[29] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "CuSha: vertex-centric graph processing on GPUs," in *HPDC*, 2014.