Accelerating Large-Scale Graph Processing with FPGAs
Lesson Learned and Future Directions

Procaccini, Marco; Sahebi, Amin; Barbone, Marco; Luk, Wayne; Gaydadjiev, Georgi; Giorgi, Roberto

DOI
10.4230/OASIcs.PARMA-DITAM.2024.6

Publication date
2024

Document Version
Final published version

Published in
15th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures - 13th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms, PARMA-DITAM 2024

Accelerating Large-Scale Graph Processing with FPGAs: Lesson Learned and Future Directions

Marco Procaccini
University of Siena, Italy

Amin Sahebi
University of Siena, Italy

Marco Barbone
Imperial College London, UK

Wayne Luk
Imperial College London, UK

Georgi Gaydadjiev
Delft University of Technology, The Netherlands

Roberto Giorgi
University of Siena, Italy

Abstract

Processing graphs on a large scale presents a range of difficulties, including irregular memory access patterns, device memory limitations, and the need for effective partitioning in distributed systems, all of which can lead to performance problems on traditional architectures such as CPUs and GPUs. To address these challenges, recent research emphasizes the use of Field-Programmable Gate Arrays (FPGAs) within distributed frameworks, harnessing the power of FPGAs in a distributed environment for accelerated graph processing. This paper examines the effectiveness of a multi-FPGA distributed architecture in combination with a partitioning system to improve data locality and reduce inter-partition communication. Utilizing Hadoop at a higher level, the framework maps the graph to the hardware, efficiently distributing pre-processed data to FPGAs. The FPGA processing engine, integrated into a cluster framework, optimizes data transfers, using offline partitioning for large-scale graph distribution. A first evaluation of the framework is based on the popular PageRank algorithm, which assigns a value to each node in a graph based on its importance. In the realm of large-scale graphs, the single FPGA solution outperformed the GPU solution that were restricted by memory capacity and surpassing CPU speedup by 26x compared to 12x. Moreover, when a single FPGA device was limited due to the size of the graph, our performance model showed that a distributed system with multiple FPGAs could increase performance by around 12x. This highlights the effectiveness of our solution for handling large datasets that surpass on-chip memory restrictions.

2012 ACM Subject Classification Hardware → Hardware accelerators

Keywords and phrases Graph processing, Distributed computing, Grid partitioning, FPGA, Accelerators

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2024.6

Related Version
Full Version: `https://link.springer.com/article/10.1186/s40537-023-00756-x`

Supplementary Material
The source code related to this paper is available from the authors upon request.

1 Introduction

Graph computing is a specialized field within computer science dedicated to the analysis and manipulation of data organized in a graph structure. This approach aims to reveal the patterns, relationships, and insights inherent in interconnected data, particularly in domains

15th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 13th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2024).
Editors: João Bispo, Sotirios Xydis, Serena Curzel, and Luís Miguel Sousa; Article No. 6; pp. 6:1–6:12
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

such as social networks, biology, and transportation [22, 21]. The use of graph structures improves the extraction of valuable information, contributing to better decision-making and system improvement.

The emergence of big data has caused an increase in the size of models, datasets, and graphs, which are known as large-scale graphs. The volume of data organized in graph structures is expanding rapidly, necessitating efficient and scalable techniques for processing these large-scale graphs [18]. The difficulties arise from dealing with the large amount of data and the complex structure of the graphs. When dealing with large-scale graphs that contain billions of nodes and edges, the demand extends beyond the raw computing power. A single node is not capable of handling such large graphs, so distributed large-scale graph computing is a beneficial solution. Furthermore, specialized methods, such as graph partitioning or optimizing random memory access, are crucial for improving efficiency and scalability in managing these complex datasets.

Field Programmable Gate Arrays (FPGAs) are becoming increasingly popular for graph processing, offering an attractive alternative to conventional CPUs and GPUs [6, 9]. Indeed, GPUs are optimized for massively parallel workloads, such as those seen in Deep Neural Networks. However, their efficiency decreases when faced with applications that involve highly memory-sparse operations and issues related to data races[17]. On the other hand, CPUs, while more general-purpose, face limitations in large-scale graph processing due to their intrinsic architecture, which is not inherently optimized for the intricate and parallel nature of graph-related computations. Unlike conventional architectures, FPGAs are remarkable for their high level of customization, providing optimized performance for specialized tasks [5, 12, 10, 11].

This work outlines the challenges and insights associated with large-scale graph processing encountered during the evaluation of our framework presented in [19]. The main objective is to evaluate and suggest improvements based on the initial examination of a structure that uses large-scale graph processing with FPGAs in a distributed system based on Hadoop.

The rest of this paper is structured as follows: In Section 2 we present the motivation and challenges in dealing with large-scale graph computing. In Section 2.1, we introduce our framework for accelerating large-scale graph computing with FPGAs. In Sections 3 and 4 we discuss the methodology of design implementation and its evaluation. Finally, Section 5 presents our conclusion and provides a brief overview of potential future directions.

2 Motivation and Challenges

Recent research on graph processing using FPGAs [5] has been conducted mainly on medium-sized data sets, rather than large-scale ones. Sakr et al. [20] have shown that these medium-sized graphs can be managed by desktop CPUs. This could reduce the attractiveness of using hardware accelerators such as FPGAs or GPUs, as they are more expensive and less user-friendly than general-purpose CPUs.

Motivations

The emergence of big data technologies has changed the landscape, making it easier to collect, store, and process large amounts of data. This has led to an abundance of data, often in the form of graphs, which have grown to the point of reaching PetaBytes [20], surpassing the memory capacity of current CPUs or GPUs. For instance, when the graph size exceeds GPU memory limits, unified memory becomes essential. This requires frequent data transfers between host memory and GPU on-chip memory, incurring additional overhead and leading to performance degradation [16].

Our second motivation is to integrate a high-level interface for the deployment of a distributed platform on the underlying hardware. Hadoop is a valuable solution for large-scale graph processing, providing powerful tools for storing, processing, and analyzing extensive graph datasets in a distributed manner. The Hadoop Distributed File System (HDFS) allows for the storage of large datasets across a cluster of machines, making it possible to process graphs that are too large for a single machine. Hadoop's scalability allows for the flexible adaptation of the cluster by adding or removing machines, making it cost-effective to process large graphs. Using the map-reduce programming model inherent in Hadoop enables distributed computing, significantly improving the efficiency of graph processing algorithms.

Challenges

When designing large-scale graph processing on FPGAs, it is essential to make critical design decisions. To begin with, it is necessary to divide the graph into small, equal-sized parts due to the limited on-chip memory in modern FPGAs. This partitioning technique is essential since FPGAs do not have the capability to do dynamic memory allocations. It is of utmost importance to reduce external memory accesses, as data transfers can cause considerable overhead and have a negative effect on performance. The design of a processing kernel is critical as it should have a memory access pattern that is compatible with the partitioning scheme mentioned above. This decision is made to reduce communication overhead, particularly when reading host computer memory from the accelerator or between different accelerators. The processing kernel should be as efficient as possible, taking advantage of the parallelism that can be achieved on FPGAs. Multiple instances of computational units can be created to increase parallelism and improve performance.

2.1 Proposed Solution

In this section, we will delve into the structure of the framework, exploring the partitioning techniques employed, the single-FPGA architecture and its adaptation for the multi-FPGA distributed system.

Graph Partitioning

Graph partitioning is a well-discussed challenge in the graph computing literature, with numerous works proposing innovative techniques and algorithms [23].

This framework utilizes the GridGraph partitioning technique to divide the edges of a graph into smaller, self-contained chunks, each of which is assigned to a particular vertex. These chunks, along with the associated vertex data, are stored on the host computer's file system. This design ensures that the chunks are independent and compatible with the Block RAM (BRAM) size of the target FPGA. During the processing, the kernel reads the chunks in sequence from the host memory, updating the values, which are then written back. GridGraph is a partitioning technique that is selected for FPGA acceleration due to its advantageous tradeoffs. It is especially beneficial for large-scale graph processing scenarios, as it divides the graph into smaller grids for independent processing, resulting in high data locality and avoiding data conflicts [23]. Furthermore, the ability to map grids onto on-chip FPGA resources increases performance scalability. An illustration of graph partitioning with GridGraph is shown in Fig. 1, which divides the vertex set into grid subsets. Each partition represents outgoing edges for a specific range of vertices, facilitating an efficient iterative processing sequence. The process loads and processes edges from each partition in sequence, computing vertex values until a specific termination condition for the algorithm is met.

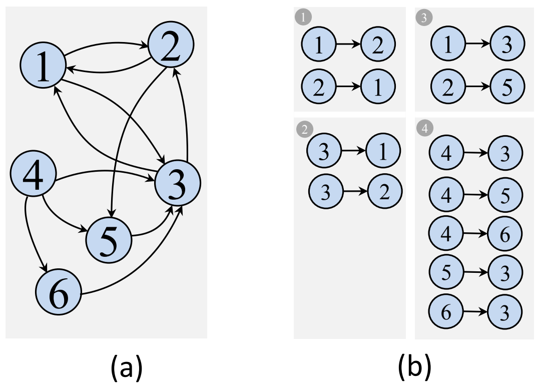

Figure 1 (a) A given sample graph. (b) Produced edge blocks using GridGraph partitioning. The number of partitions is P=2, producing P^2 edge blocks.

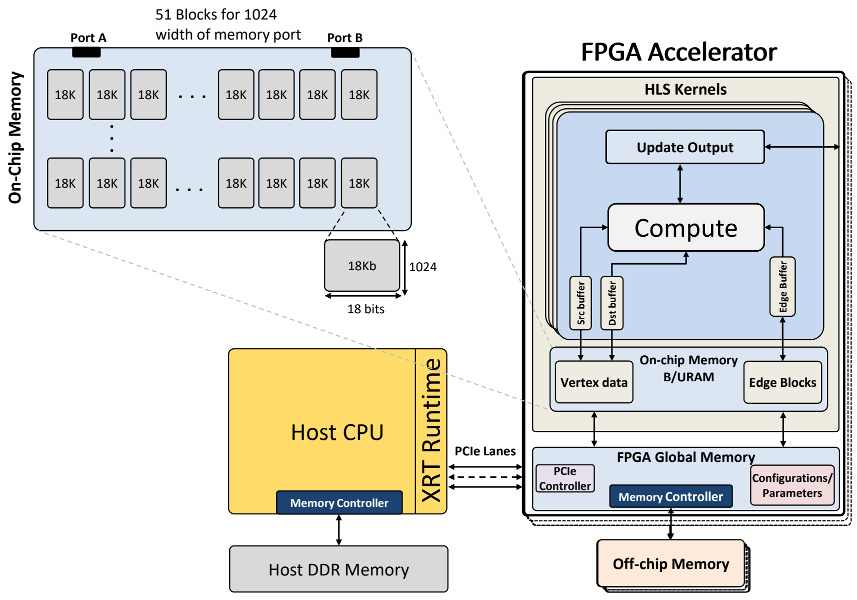

Figure 2 An overview block diagram of the proposed hardware design [19].

FPGA Architecture

In a single FPGA process, the graph data is pre-processed using the partitioning technique described in Section 2.1. The resulting edge blocks are then stored on the host file system. Given that the overall size of these graph blocks may reach terabytes, the host memory must be large enough to accommodate the reading of all edge blocks from the file system. Subsequently, the data is loaded into the FPGA on-chip memory before processing. Fig. 2 provides an overview of the single FPGA graph processing unit. In this representation, the on-chip memory is configured to optimize the memory bandwidth while maintaining the system frequency at its peak. Dedicated FPGA kernels are responsible for reading the data from the stream input provided by the host and directing them to the computational units within the FPGA. The units execute the graph algorithm, such as PageRank, afterwards. Upon completion of the computation, the aggregated results are written back to the host memory and stored in its file system.

Distributed Architecture

Graph datasets that are too large to be processed on a single machine can be handled in distributed computing. This involves distributing the data across multiple machines, allowing for the processing of graphs that would be too large for a single machine to handle. A distributed system is made up of multiple machines that work together as one virtual system. Each machine or node is responsible for processing a portion of the data. These systems can be managed manually with custom software, such as popular message passing interfaces (MPI). This manual management allows for precise application optimization, leading to high performance. However, this approach requires a lot of engineering effort and knowledge, especially in the field of distributed computing. Various distributed computing frameworks are available that automate and tackle many of the issues discussed earlier. Hadoop, a popular open source framework created by the Apache Software Foundation in 2005, is a widely used technology for large-scale data processing in machine clusters [2]. It is based on the map-reduce programming model and enables parallel processing of large amounts of data across a distributed platform, due to the integration of the Hadoop Distributed File System (HDFS). Data are usually stored in a distributed file system, such as HDFS or ZFS, and processed through a distributed computing framework, such as Apache Hadoop. Graph algorithms, including PageRank, can be implemented in these frameworks for graph processing and analysis [8]. Recently, the integration of FPGAs with Hadoop for large graph processing has gained attention [3]. This combination allows one to take advantage of Hadoop's scalability and fault tolerance while taking advantage of the high performance of FPGAs for graph processing tasks. Although the use of FPGAs with Hadoop is still in its early stages, further research is needed to evaluate the feasibility of using FPGAs in combination with Hadoop to speed up graph processing and optimize system performance.

Data processing within the map-reduce architecture involves two main phases. In the initial "Map" phase, the data are divided into smaller chunks known as input splits. Each split is processed by a separate node in the cluster, utilizing user-defined functions called Mappers to transform the input data into intermediate key-value pairs. The subsequent "Reduce" phase processes these intermediate key-value pairs using user-defined functions called Reducers, which merge the input data into the final output. In a map-reduce system, a user application initiates a root controller and a set of mappers and reducers distributed across various compute nodes. The root node coordinates the generation of mappers and reducers and monitors their progress. The overall system overview of the Hadoop map-reduce design is depicted in Fig. 3.

In our scenario, nodes with multiple FPGA accelerators are configured to appear as multiple nodes with a single FPGA to Hadoop. For instance, a node with four FPGAs executes four distinct instances of Hadoop, each FPGA being mapped one-to-one to an instance. This design simplifies the distribution of workload across multiple FPGAs, eliminating the need for manual splitting. Moreover, it allows for the use of Hadoop scheduling for load balancing and fault tolerance. The Hadoop scheduler can handle single FPGA failures without taking the entire node offline. The framework for large-scale graph processing involves three primary phases (see Fig. 3). During the initial stage, the graph is divided into smaller sections, named edge-blocks, using the GridGraph partitioning technique. This initial step serves as pre-processing, enabling the parallel processing of sub-graphs through a single scan. The second phase constitutes the core processing stage, where a customized Map function executes the graph processing algorithm on the previously computed subgraphs (e.g., PageRank). The third and final phase involves the integration of partial results generated by different workers using a custom Reduce function. This step is optional and can be utilized to merge results

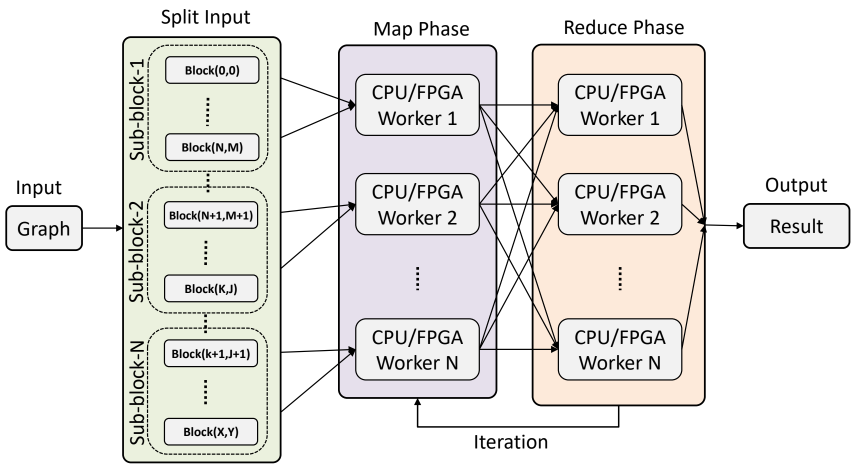

Figure 3 The Hadoop framework for distributed graph processing [19].

saved in different files by Mappers. However, the necessity of consolidating results into a single monolithic file may vary depending on the specific use case. For iterative algorithms, such as PageRank, it is necessary to repeat phases 2 and 3 until the algorithm converges.

3 Methodology

In this section, our aim is to provide insight into the methodology employed in developing the graph processing framework. This framework is implemented using the Vivado HLS toolchain (version 2022.2) on the Xilinx Alveo U250 board, featuring the Xilinx VU13P FPGA. Xilinx Alveo boards serve as Data Center accelerator cards tailored for diverse tasks such as machine learning inference, video transcoding, and database search and analytics. The design utilizes the Vivado design suite (Vitis version 2022.2) and is divided into two parts: implementation of the kernel and host program. The host program, executed on the CPU, comprises two main stages. The initial stage involves receiving and preparing pre-processed graph data (see Section 2.1) and parameters, while the subsequent step creates buffers for optimal memory alignment to enhance performance. The bridge between the host and the kernels is established through runtime buffers and commands to program the FPGA device with the bitstream. Data exchange between host memory and kernel local memory, like BRAM, is facilitated by using OpenCL functions and specific FPGA kernels.

On the contrary, the kernel implementation, executed on the FPGA, is written in HLS and structured to optimally utilize accelerator resources. Efficient FPGA implementation is influenced by structures such as local arrays, which require careful resource allocation in terms of Lookup Tables (LUTs), Block RAM (BRAM), and registers. Strategies such as partitioning and streaming data through small and fast FIFOs are employed to minimize resource utilization and accommodate large local arrays on FPGAs. To enhance performance, the proposed design incorporates tuning at the kernel link stage, allowing a single kernel to instantiate multiple hardware compute units (CUs). The host program initiates overlapping kernel calls, executing kernels concurrently by running independent CUs.

In this paper, we present the performance evaluation of our framework in the single FPGA configuration, extracting the execution time of the FPGA kernel. For distributed execution, since there is no access to a Hadoop cluster, we employed a mathematical model to estimate performance. The map-reduce programming model has been extensively explored in the

Table 1 The XACC server is used to evaluate the real implementation.

Instance Name	CPU	CPU Freq	Cores	Memory	FPGA board
alveo1.ethz.ch	2× Intel Xeon Gold 6234	3.30 GHz	16	376 GiB	Alveo U250

literature, allowing for predictable performance modeling [7, 13]. Consequently, this study relies on performance forecasts drawn from previous analyses of algorithm efficiency in map-reduce implementations. Before delving into the analysis, it is crucial to underline specific assumptions. First, the graph is stored in the Hadoop Distributed File System (HDFS), and second, graph partitioning occurs in parallel through a MapReduce job or custom partitioner. The first assumption remains valid, given the focus on large-scale graphs, which are too extensive for a single node, making it likely that they are stored in a distributed file system. The second assumption is valid as long as the first holds. Once the graph is situated in the file system, any necessary pre-processing can be directly executed in MapReduce. Given these assumptions, the graph scale necessitates analysis in a distributed system, as attempting such an analysis on a single parallel node would require an impractical amount of time. In the distributed framework, FPGA accelerators are exclusive to mappers for computing a partial state. The Eq. (1) provided is essential for assessing how a faster mapper would impact the overall computation time.

$$T = T_{split-input} + T_{overhead} + N(T_{scatter} + max(T_{map}) + T_{transfer} + T_{reduce}) \tag{1}$$

where:

- $T_{split-input}$ time needed to partition the graph in sub-graphs;
- N is the number of iterations of the iterative algorithm;
- $T_{overhead}$ is the overhead introduced by map-reduce;
- $T_{scatter}$ is the time needed to distribute the state vector to all the workers;
- $max(T_{map})$ is the time needed by the slowest mapper;
- $T_{transfer}$ is the time needed to transfer the state vector to the reducers (account for shuffle and sort);
- T_{reduce} is the time needed to aggregate the partial state vectors.

4 Evaluation

This section presents the first evaluation of the framework using an optimized version of the PageRank algorithm. PageRank is often used as a standard for evaluating the performance of large graph processing. It is one of the most computationally intensive graph algorithms and requires the processing of a large number of vertices and edges. This algorithm is used to assess the importance of each node in a graph, which was initially used by search engines to rank webpages based on their relevance. PageRank assigns a score, referred to as the PageRank score, to each vertex, based on the number and importance of the vertices pointing to it. Vertices with higher PageRank scores are considered more significant than those with lower scores. Using PageRank as a measure allows for the assessment of the proposed model's effectiveness in dealing with large graph data efficiently.

Table 2 The datasets for evaluating our proposed study. We choose them based on the size and the structure of the datasets to be comparable with other works.

Graph dataset	Vertices	Edges	Size (GB)	Type
LiveJournal [15]	4.8M	0.069 B	1.1	Social Web
Web-UK-2005 [4]	39M	0.994 B	16	Web Graph
Twitter [14]	41.6 M	1.47 B	23	Web Graph
Friendster [15]	68.3M	2.58 B	43	Social Web

The implementation of the framework described in section 2.1 has been evaluated against CPU, GPU, and FPGA architectures. The hardware implementation was carried out on a Xilinx Alveo U250, which was integrated into Xilinx Adaptive Compute Clusters (XACC) [1]. The XACC servers distributed resources evenly across multiple Virtual Machines (VMs), guaranteeing that each VM had access to its own FPGA card. The software environment within the VM is based on Ubuntu 20.04 and incorporates FPGA accelerator deployment frameworks such as Vitis and Vivado HLS. Table 1 provides detailed specifications for the server and the hardware accelerator.

In the context of PageRank comparisons with the CPU, we used both a sequential version, an OpenMP multicore version, and the GridGraph library, recognized as one of the most efficient graph processing frameworks for CPUs. The OpenMP version used in this assessment is version 4.0.3, and the software is compiled using GCC version 9.4.0. In the case of GPU implementation, we used the cuGraph library for comparison, using the CUDA toolkit version 11.7. "cuGraph" is an open source GPU graph analysis library integrated into the RAPIDS ecosystem, offering a high-performance, user-friendly, and extensible framework for GPU-based graph analysis. Our experiments used the NVIDIA V100 GPU, renowned for its high performance based on the Volta architecture. The cugraph library was chosen not only for its capabilities in single-GPU execution, but also for its potential in running multi-GPU executions to facilitate a comprehensive comparison with our distributed execution. Nevertheless, it is important to mention that our attempts to execute the multi-GPU function with our extensive dataset were unsuccessful, indicating the need for additional effort to resolve the issue. To evaluate our optimized PageRank algorithm, we selected graphs of various sizes from the datasets listed in Table 2, ranging from a small graph (LiveJournal) to a large graph (Friendster). These datasets were intentionally chosen to assess performance as the graph size exceeds the memory capacities of the CPU or GPU devices.

Fig. 4 shows that the FPGA-based PageRank implementation outperforms sequential and OpenMP execution on a CPU for all datasets. The speedup achieved ranges from approximately 9.7x for the smallest dataset to about 26x for the Twitter dataset. Compared to the GridGraph library, which employs a grid partition schema similar to ours and uses OpenMP for parallel execution, the framework shows a performance improvement with a speedup up to 1.5x (Twitter). Our FPGA implementation not only overcomes CPU solutions but also achieved a speedup of up to 4.5x compared to the cuGraph library (Web-UK-2005). In particular, the Web-UK-2005 data set, exceeding GPU on-chip memory, requires the use of Unified Memory, incurring additional overhead and degrading performance due to data transfers between host memory and GPU on-chip memory. Larger datasets (Twitter and Friendster) cause GPU experiments to fail due to insufficient memory on the GPU board.

Table 3 Alveo U250 platform experimental results in comparison to a software baseline and state-of-the-art FPGA works. Reported numbers are all in Seconds.

Dataset	Sequential CPU[1)]	OpenMP CPU[1)]	GridGraph CPU[1)]	cuGraph GPU[2)]	Vitis Library FPGA	Our work FPGA
LiveJournal	27.01	5.49	3.54	5.28	79.79	2.78
Web-UK-2005	275.44	185.4	34.9	90.73	N/A[3)]	20.6
Twitter	1443	658.5	88.5	Failed [4)]	N/A[3)]	55.6
Friendster	2258	950	141	Failed [4)]	N/A[3)]	95.4

1) CPU details are described in Table 1.
2) The GPU used for the experiments is a NVIDIA Volta V100.
3) N/A indicates that the aforementioned study did not report this dataset evaluation.
4) The experiment hit the GPU memory limit.

These results underscore the advantages of employing FPGA for graph processing tasks, especially with large datasets. Furthermore, the framework surpasses the PageRank algorithm available in the Vitis Library, achieving a speedup of about 28x. This emphasizes the benefits of custom FPGA implementations for graph processing tasks, particularly with large datasets that surpass on-chip memory capacity. The findings position our FPGA implementation as a suitable solution for accelerating graph processing tasks.

It is noteworthy that the framework's speedup nearly doubles from Web-UK-2005 to Twitter, despite Twitter's size being approximately 1.5 times larger than Web-UK-2005 (see Table 2). However, the performance boost diminishes with the larger Friendster dataset, prompting an exploration of whether a multi-FPGA solution in a distributed system, like Hadoop, could enhance FPGA usage for large-scale graph processing.

Graphs larger than Friendster exceed the computing capabilities of contemporary machines. To address this, a potential solution involves distributing these large graphs across multiple machines through distributed computing. Fig. 5 presents forecasts for the integration of FPGA acceleration in a Hadoop-distributed cluster for large-scale graph processing. Profiling the code and using the results to evaluate Eq. (1) indicates that FPGA utilization is most effective when a significant amount of time is spent in the mapping phase. In the forecasts, it is assumed that the majority of time (51% of the total time) is spent in the mapping phase. Under these conditions and considering a worst-case scenario in which FPGA-accelerated nodes achieve only a 20x speedup compared to CPU-only nodes, the forecasts show a potential 54% reduction in total time. In a more realistic scenario where 80% of the time is spent in the mapping phase (a common assumption with partition methods such as GridGraph to minimize data transfer), the time reduction achieved by a hybrid CPU-FPGA Hadoop cluster can increase to 84% compared to a CPU-only cluster. Fig. 5 summarizes these forecasts and illustrates the best-case scenario (although unlikely) where 90% of the total execution time is spent in the mapping phase, showing a potential over 90% time reduction under optimal conditions.

This initial evaluation has revealed the current performance of the framework, as well as providing useful knowledge that can be used to improve it. Examining the results has exposed areas where the framework can be improved and optimized. We investigated the graph partitioning technique and discovered that the first possible enhancements can be achieved by refining the graph partitioning approach described in Section 2.1. As observed

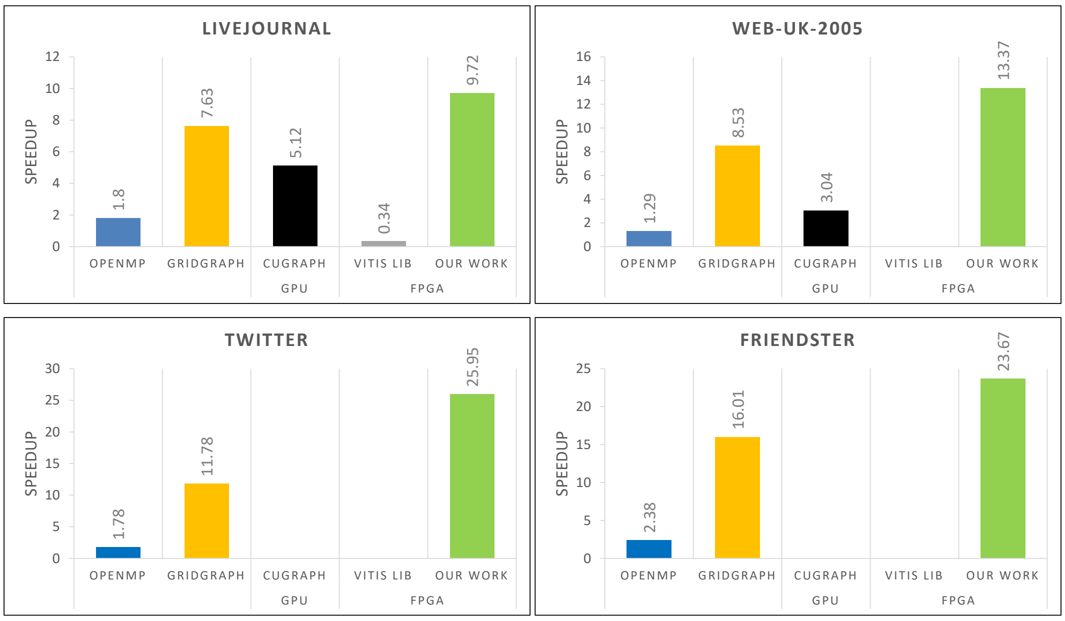

Figure 4 Speedup evaluation on the sequential execution of the proposed FPGA PageRank algorithm (our work) with the CPU, GPU, and FPGA solutions for the LiveJournal, Web-UK-2005, Twitter, and Friendstser datasets [19].

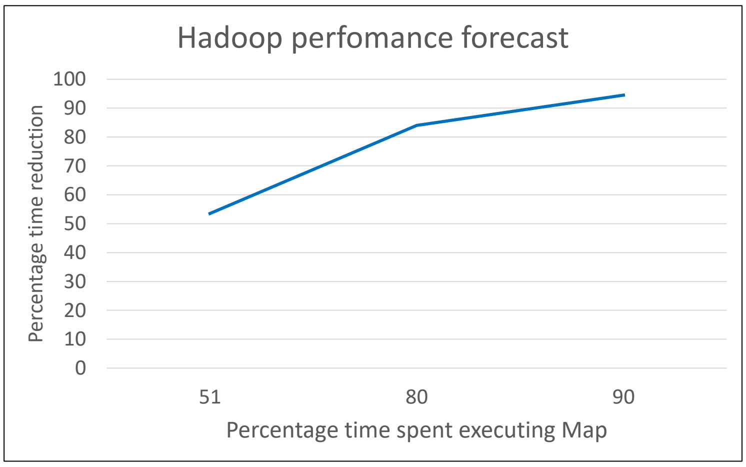

Figure 5 Reduced processing time when employing FPGAs for accelerated graph processing in contrast to utilizing a Hadoop cluster without FPGA integration [19].

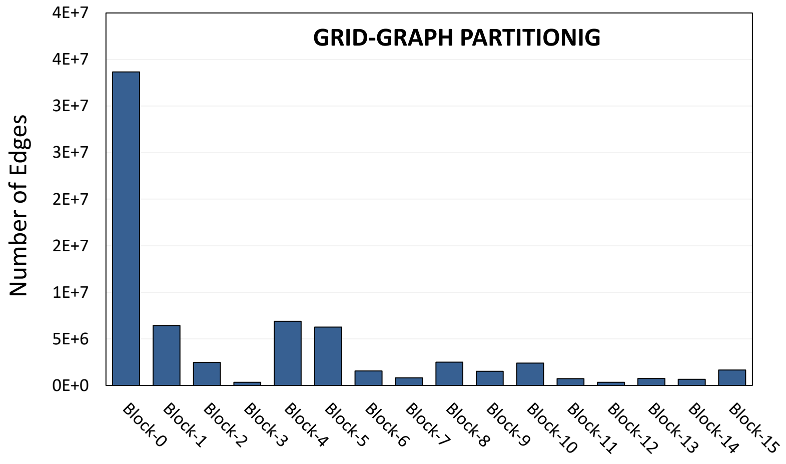

Figure 6 Edge block partitioning of the LiveJournal dataset with P=4 partitions using the GridGraph method.

in Fig. 1, the partitions are unevenly balanced in the grid, with partition 4 being larger than the others. Our investigation extended to the LiveJournal dataset, where the execution of the grid graph partitioning exhibited a similar imbalance, as illustrated in Fig. 6. The output of the grid graph resulted in one substantial partition (block 0) and several smaller partitions. This lack of balance in workload could lead to notable performance declines. Addressing this issue requires the development of an improved version of the graph partitioning method to ensure a more balanced workload, which can significantly improve overall performance.

5 Conclusions

The field of large graph processing is growing in importance as the amount of data generated by applications such as social networks, web graphs, and biological networks continues to increase. To handle this increase in graph sizes, efficient and scalable processing methods must be developed. The integration of Field Programmable Gate Arrays (FPGAs) with the open source Hadoop framework is becoming increasingly popular for the purpose of managing large graph datasets. FPGAs, which are integrated circuits designed for specific tasks, are highly efficient when it comes to executing graph processing algorithms. Hadoop, on the other hand, is a widely used platform for distributed processing of large datasets. The first evaluation of our framework has been shown to be more effective than existing implementations for the PageRank algorithm. This highlights the usefulness of FPGA-based solutions for large datasets. Additionally, the combination of FPGAs and Hadoop can potentially improve performance when dealing with large datasets. This initial evaluation highlights the potential improvements of the framework, stressing the importance of taking into account factors such as graph partitioning. Further research is needed to broaden the benchmark set for a more comprehensive evaluation and to look into other essential aspects such as the host-FPGA communication and the FPGA memory usage.

References

1 AMD Xilinx. Heterogeneous Accelerated Compute Clusters. `https://www.amd-haccs.io/`, 2023 (accessed 15 November 2023).
2 Apache Hadoop. Hadoop. Accessed 30 Jan 2023. URL: `https://hadoop.apache.org/`.

3 Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, et al. The future of FPGA acceleration in datacenters and the cloud. *ACM TRETS*, 15(3):1–42, 2022.

4 Paolo Boldi and Sebastiano Vigna. The web graph framework I: Compression techniques. In *Proc. of the 13th ACM International WWW Conference*, pages 595–601, NY, USA, 2004.

5 Xinyu Chen, Hongshi Tan, Yao Chen, Bingsheng He, Weng-Fai Wong, and Deming Chen. ThunderGP: HLS-based graph processing framework on FPGAs. In *The ACM/SIGDA International Symposium on FPGAs*, FPGA '21, pages 69–80, New York, NY, USA, 2021.

6 Guohao Dai, Tianhao Huang, Yuze Chi, Ningyi Xu, Yu Wang, and Huazhong Yang. ForeGraph: Exploring large-scale graph processing on multi-FPGA architecture. In *Proc. of the 2017 ACM/SIGDA International Symposium on FPGAs*, FPGA '17, page 217226, NY, USA, 2017.

7 Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

8 Benedikt Elser and Alberto Montresor. An evaluation study of BigData frameworks for graph processing. In *2013 IEEE International Conference on Big Data*, pages 60–67, 2013.

9 Nina Engelhardt and Hayden K.-H. So. GraVF-M: Graph processing system generation for Multi-FPGA platforms. *ACM Trans. Reconfigurable Technol. Syst.*, 12(4), November 2019.

10 Antonio Filgueras, Miquel Vidal, Marc Mateu, Daniel Jiménez-González, Carlos Álvarez, Xavier Martorell, Eduard Ayguadé, Dimitrios Theodoropoulos, Dionisios Pnevmatikatos, Paolo Gai, Stefano Garzarella, David Oro, Javier Hernando, Nicola Bettin, Alberto Pomella, Marco Procaccini, and Roberto Giorgi. The axiom project: Iot on heterogeneous embedded platforms. *IEEE Design Test*, pages 1–1, 2019.

11 R. Giorgi, F. Khalili, and M. Procaccini. AXIOM: A Scalable, Efficient and Reconfigurable Embedded Platform. In *IEEE Proc.DATE*, pages 1–6, March 2019.

12 R. Giorgi, Farnam. Khalili, and Marco Procaccini. Translating Timing into an Architecture: The Synergy of COTSon and HLS. *Hindawi – IJRC*, 2019:1–18, December 2019.

13 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM, 2010.

14 Jérôme Kunegis. Konect: The koblenz network collection. In *Proc. of the 22nd Int. Conf. on WWW*, pages 1343–1350, New York, NY, USA, 2013. ACM.

15 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, 2014. URL: `http://snap.stanford.edu/data`, (accessed 15 November 2023).

16 Wenqiang Li, Guanghao Jin, Xuewen Cui, and Simon See. An evaluation of unified memory technology on NVIDIA GPUs. In *2015 15th IEEE/ACM international symposium on cluster, cloud and grid computing*, pages 1092–1098. IEEE, 2015.

17 Yashuai Lü, Hui Guo, Libo Huang, Qi Yu, Li Shen, Nong Xiao, and Zhiying Wang. GraphPEG: Accelerating graph processing on GPUs. *ACM TACO*, 18(3):1–24, 2021.

18 M. Usman Nisar, Arash Fard, and John A. Miller. Techniques for graph analytics on big data. In *2013 IEEE International Congress on Big Data*, pages 255–262, 2013.

19 Amin Sahebi, Marco Barbone, Marco Procaccini, Wayne Luk, Georgi Gaydadjiev, and Roberto Giorgi. Distributed large-scale graph processing on fpgas. *Journal of Big Data*, 10(1):95, 2023.

20 Sherif Sakr and et al. Bonifati. The future is big graphs: A community view on graph processing systems. *Commun. ACM*, 64(9):62–71, August 2021.

21 Justin S Smith, Adrian E Roitberg, and Olexandr Isayev. Transforming computational drug discovery with machine learning and AI, 2018.

22 Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. on NNLS*, 32(1):4–24, 2021.

23 Xiaowei Zhu, Wentao Han, and Wenguang Chen. GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning. In *Proc. of the 2015 Conf. on Usenix Annual Technical Conference*, pages 375–386, USA, 2015.