electronics

MDPI

Article

Performance Evaluation of Parallel Graphs Algorithms Utilizing Graphcore IPU

Paweł Gepner [1,*], Bartłomiej Kocot [2], Marcin Paprzycki [3], Maria Ganzha [3], Leonid Moroz [1] and Tomasz Olas [4]

[1] Faculty of Mechanical and Industrial Engineering, Warsaw University of Technology, Narbutta 86, 02-524 Warszawa, Poland; leonid.moroz@pw.edu.pl
[2] Centre of Informatics—Tricity Academic Supercomputer & Network (CI TASK), Gdansk University of Technology, 80-233 Gdańsk, Poland; bartlomiej.kocot@pg.edu.pl
[3] Systems Research Institute, Newelska 6, 01-447 Warszawa, Poland; marcin.paprzycki@ibspan.waw.pl (M.P.); maria.ganzha@ibspan.waw.pl (M.G.)
[4] Faculty of Computer Science and Artificial Intelligence, Czestochowa University of Technology, Dąbrowskiego 73, 42-200 Czestochowa, Poland; olas@icis.pcz.pl
* Correspondence: pawel.gepner@pw.edu.pl

Abstract: Recent years have been characterized by increasing interest in graph computations. This trend can be related to the large number of potential application areas. Moreover, increasing computational capabilities of modern computers allowed turning theory of graph algorithms into explorations of best methods for their actual realization. These factors, in turn, brought about ideas like creation of a hardware component dedicated to graph computation; i.e., the Graphcore Intelligent Processor Unit (IPU). Interestingly, Graphcore systems are a hardware implementation of the Bulk Synchronous Parallel paradigm, which seemed to be a mostly theoretical concept from the end of last century. In this context, the question that has to be addressed experimentally is as follows: how good are Graphcore systems in comparison with standard systems that can be used to run graph algorithms, i.e., CPUs and GPUs. To provide a partial response to this broad question, in this contribution, PageRank, Single Source Shortest Path and Breadth-First Search algorithms are used to compare the performance of IPU-deployed algorithms to other parallel architectures. Obtained results clearly show that the Graphcore IPU outperforms other devices for the studied heterogeneous algorithms and, currently, provides best-in-class execution time results for a range of graph sizes and densities.

Keywords: intelligent processor unit; performance evaluation; parallel graph algorithms

check for updates

Citation: Gepner, P.; Kocot, B.; Paprzycki, M.; Ganzha, M.; Moroz, L.; Olas, T. Performance Evaluation of Parallel Graphs Algorithms Utilizing Graphcore IPU. *Electronics* **2024**, *13*, 2011. https://doi.org/10.3390/electronics13112011

Academic Editors: Seongah Jeong, Jin-Hyun Ahn and Jin-kyu Kang

Received: 3 April 2024
Revised: 10 May 2024
Accepted: 16 May 2024
Published: 21 May 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Graph representation, and its associated algorithms, have permeated numerous domains, addressing a wide array of complex challenges. From optimizing public transport routes to analyzing social networks and mapping neural connections in medicine, graphs serve as versatile tools for modeling and understanding complex systems [1–3]. However, as the scale and complexity of these applications continue to grow, traditional sequential processing methods struggle to keep pace, necessitating the adoption of parallel graph algorithms.

In the financial sector, for instance, where graphs are employed to detect fraudulent activities and monitor financial flows [4,5], the sheer volume of transactions demands efficient parallel processing solutions. Similarly, within manufacturing contexts, where graphs help manage dependencies between production components and machines, the need for parallel algorithms becomes evident [6]. Graph-based analyses enable the optimization of workflows and resource utilization, but their effectiveness hinges on the ability to process vast amounts of data in parallel.

Moreover, in law enforcement and governance, the use of graph algorithms to combat crime underscores the importance of parallel processing capabilities [7,8]. Identifying connections within organized crime networks or tracking tax evasion schemes requires the rapid analysis of large-scale graph data, a task well suited for parallel computing architectures.

In marketing, where personalized targeting is paramount, graph-based analyses facilitate the extraction of meaningful insights from the extensive customer data [9]. Parallel processing enables marketers to efficiently match products with specific customer profiles, enhancing the effectiveness of targeted advertising campaigns.

Despite their utility, many graph algorithms pose significant computational challenges, particularly when dealing with large datasets. Traditional CPU-based approaches struggle to meet the performance demands imposed by these algorithms. Similarly, due to the complex non-structured nature of graph-represented data, GPU-based approaches are not easy to efficiently realize. Here, it is also important to recall that each "input graph" may be very different from other inputs. Hence, practical use of FPGA-based realization of graph algorithms brings more questions than answers. This all prompts exploration of alternative solutions. Among them, Graphcore's IPU presents a compelling option, offering the parallel processing power necessary to accelerate graph analytics tasks [10]. In this context, the aim of this work is to explore practical aspects of running three (heterogeneous) graph algorithms on a Graphcore IPU-based system and compare the obtained performance with that of typical modern parallel computers based on CPUs and GPUs.

Before proceeding, let us make a few methodological comments. (1) The aim of this work is to further establish that the Bulk Synchronous Parallelism (BSP) approach to realization of graph algorithms on Graphcore IPU is competitive to standard modern approaches. This being the case, none of the three algorithms have been fine-tuned to achieve the best possible results on the system they have been run on. (2) Since, in general, practically realizable graph algorithms have high computational complexity, this is the only factor that is measured in performed experiments. It is assumed (and this was observed in practice) that for graphs that are large enough, times of algorithm compilation and problem staging are negligible for all architectures. (3) Since the algorithms are being realized using the BSP model, one of the important factors that introduces variation to performance of parallel algorithms, i.e., "thread asynchronicity", has been eliminated. Therefore, while all experiments have been run multiple times, no significant differences between execution times have been observed. (4) The three algorithms have been selected on the basis of their heterogeneity, i.e., the fact that each one of them explores different aspect of graph computations. Obviously, since in each of these cases, the Graphcore IPU turned out to be (highly) competitive, vis-a-vis standard approaches, further explorations make sense. Such explorations may involve, among others, other classes of algorithms, fine implementation tuning, etc.

Keeping this in mind, the remaining part of this text has been organized as follows. In Section 2, the pertinent state of the art has been outlined. Next, in Section 3, a summary of information related to the Graphcore IPU architecture has been presented. This is followed, in Section 4, by the description of the three algorithms. The next section, Section 5, presents and analyzes the experimental results. It is followed by a short section, Section 6, which summarize the main insights and contributions. Finally, Section 7 contains concluding remarks and proposed future research directions.

2. Literature Overview

Processing graphs with millions of nodes and edges is a popular research topic. To accelerate execution, multiple software and hardware approaches, focused on a specific aspect of graph processing, have been proposed. Let us summarize the key results reported in the pertinent literature.

One of the common techniques to accelerate graph processing is code parallelization. Currently, supercomputers have millions of threads [11]. However, using them requires development of (completely) new algorithms. Moreover, lack of portability between

different multi-core devices has to be considered. Here, even in the case of the same manufacturer, the software often has to be recompiled, or reprogrammed, to accommodate new hardware features. Separately, utterly different hardware requires different algorithm optimization. Nevertheless, as research and practice have shown, parallelization enables dealing with huge graphs.

Acceleration of graph algorithms has been considered for Nvidia GPU, FPGA, CPU, Graphcore IPU and heterogeneous systems. Speeding up graph algorithms using the CUDA parallel technique was discussed in [12–16]. Research concerning the application of FPGAs to such graph algorithms as Single Shortest Path, Weakly Connected Component Minimum Spanning Tree PageRank and others has been reported in [17–22]. Results of use of heterogeneous systems, for parallelization of large graph processing, can be found in [20,23,24]. Finally, speeding multiple graph algorithms on CPUs was summarized in [25].

In addition to algorithm optimization, for currently existing devices, there are also attempts to create a special hardware for graph processing (see [26–28]). At present, however, such devices are not being produced but only considered theoretically. The key disadvantage of this approach is its inefficiency for "standard problems". Moreover, it also requires an entirely different software stack.

In this context, the potential of Graphcore IPU, a relatively new accelerator, is insufficiently researched. So far, only the Breadth-First Search (BFS) algorithm, described in [29], has been evaluated on the IPU. However, its performance was compared with an "older" Nvidia V100 GPU. This lack of efficiency studies was the primary motivation for this contribution. This was also the reason to provide a rather extensive overview of the Graphcore IPU architecture, to provide an appropriate level of background knowledge.

3. Graphcore IPU and Platform Architecture

In Figure 1, we present a simplified depiction of the IPU die. In essence, Graphcore IPUs are characterized as distributed memory, highly parallel, multiple-instruction multiple-data (MIMD) devices. Each IPU comprises 1472 cores, accompanied by dedicated 624KiB SRAM memory per core, collectively referred to as a "tile". The tile Instruction Set Architecture (ISA) [10] incorporates specialized hardware components, such as Accumulating Matrix Product (AMP) and Slim Convolution (SLIC) units, facilitating the execution of up to 64 multiply-add instructions per clock cycle. Notably, the IPU supports both 32-bit single-precision floating-point FP32-IEEE standard and FP16-IEEE 16-bit half-precision data format, supplemented by hardware stochastic rounding support. Additionally, hardware resources include instructions for random number generation and specific transcendental operations commonly utilized in machine learning tasks.

Each tile operates six hardware execution threads, employing a time-sliced round-robin schedule to mitigate instruction and memory latency. Through this mechanism, most instructions, including memory access and vectorized floating-point operations, are completed within a single thread cycle (equivalent to 6 clock cycles). Each thread represents an entirely independent program, without constraints on group execution or lockstep program execution across threads, thereby ensuring high SRAM bandwidth [10,30].

With a total of 1472 tiles, the IPU possesses approximately 900 MB of memory, wherein only local memory is directly accessible by tile instructions, accommodating both code and data. Inter-tile shared memory access is unavailable, with each tile utilizing a contiguous unsigned 21-bit address space, commencing at address 0×0. Communication between tiles is facilitated through message passing, utilizing an all-to-all high-bandwidth exchange (theoretical, 8 TB/s). Notably, the memory boasts very low latency (6 cycles) and ultra-high bandwidth (theoretical, 47.5 TB/s), with the chip constructed from 59.4 billion transistors utilizing TSMC 7 nm manufacturing [10].

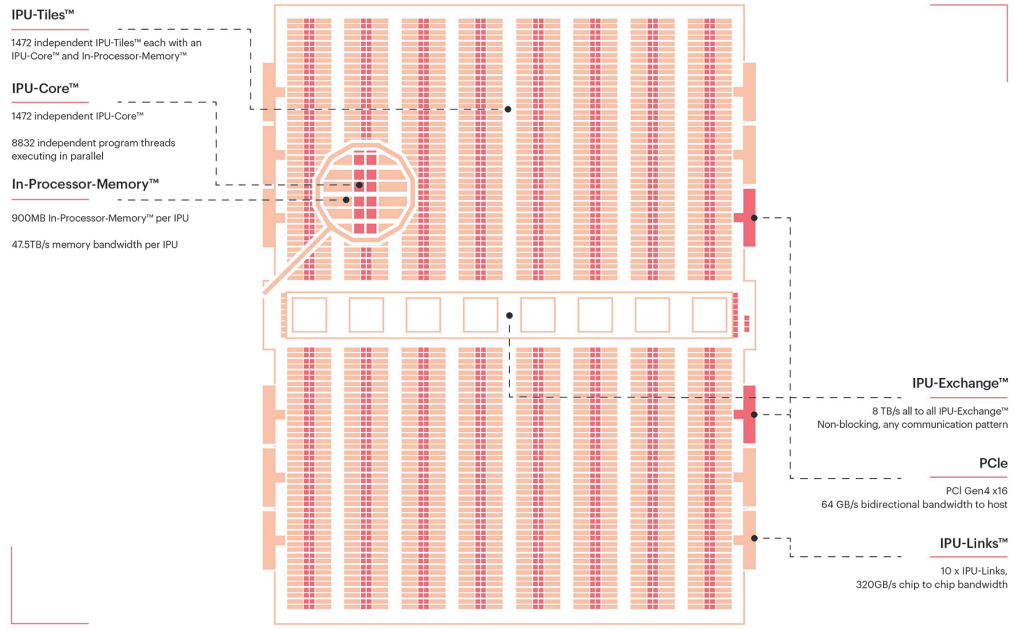


Figure 1. Simplified version of Graphcore IPU die.

The programming interface for graph-based computations is denoted as Poplar, complemented by PopLibs libraries, extending the functionality of C++ to align with the IPU operation model. In this paradigm:

- Vertices represent programs executed on individual tiles, defining operations integrated into the computation graph. The functionality of a vertex ranges from simple arithmetic operations to complex tensor data reshaping or intricate code execution.
- Computation graph delineates the input–output relationship between variables and operations, offering capabilities for construction, compilation and serialization of the computation graph.
- Control programs oversee argument administration, device selection and execution control of graph operations.

Figure 2 elucidates the concept of the IPU computation graph, elucidating the input–output relationship between variables and operations.

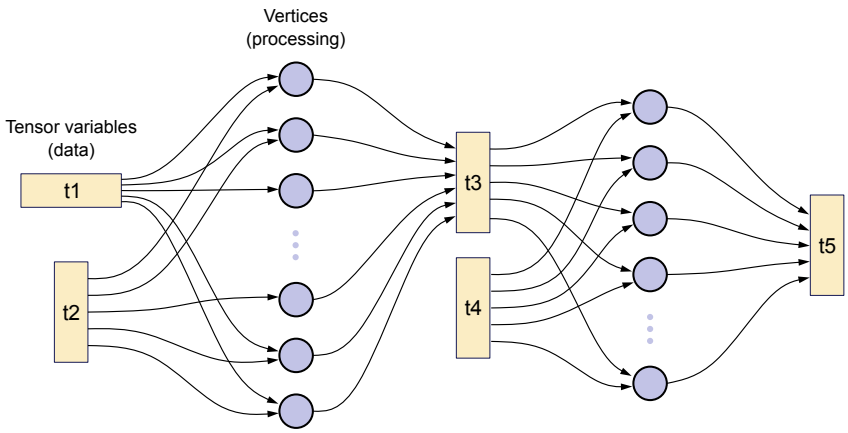


Figure 2. Concept of IPU computation graphs.

The graph encompasses tensor variables, compute tasks (vertices) and connecting edges. Data within the graph are stored in fixed-size multidimensional tensors. A vertex serves as a work fragment, its operation influenced by connecting edges determining

processed variable elements. Codelets, implemented in standard C++, are associated with each vertex, defining inputs, outputs and internal state [31].

The control program orchestrates vertex execution, managing device selection, graph loading into IPUs and program execution. Notably, data transfer between the IPU and the host, memory structures, and initiation of transfers are integral aspects of control program operation. Post deployment, all requisite code and data structures reside within the IPU's distributed memory [31].

The utilized Graphcore IPU-M2000 system, detailed in subsequent experiments, comprises four IPUs. This setup includes a gateway chip facilitating IPU interconnection and DRAM access, two 100 Gbps IPU-Fabric Links, a PCIe slot for standard Smart NICs, two 1GbE Open BMC management interfaces, and an M.2 slot. Figure 3 illustrates the block diagram of the IPU-M2000 system.

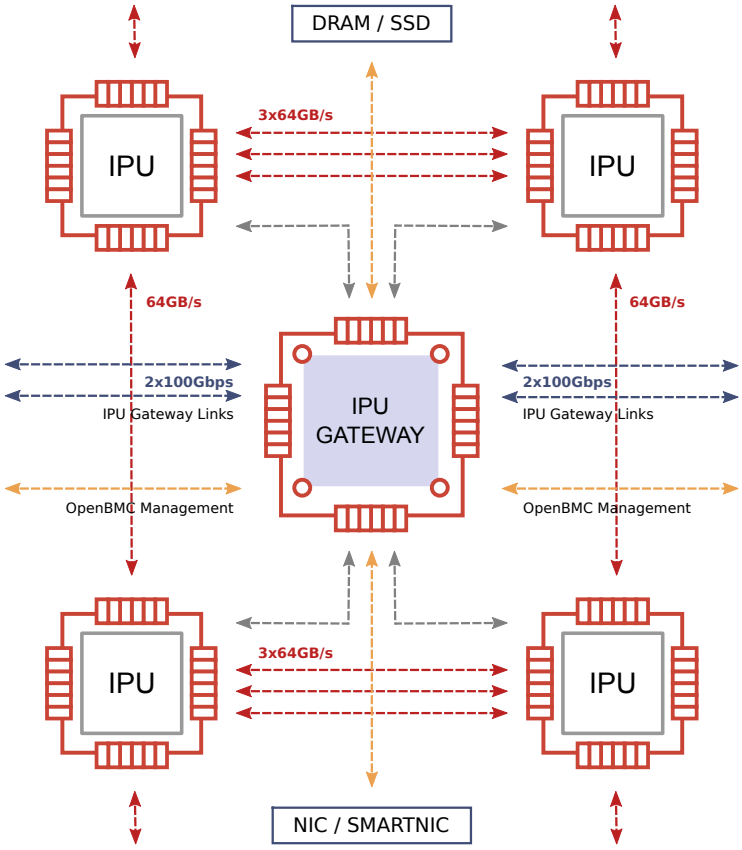


Figure 3. Schematic representation of IPU-M2000 machine.

The host system interfaces with the IPU-M2000 platform via 100Gb Ethernet, employing remote direct memory access (RDMA) over Converged Ethernet (ROCE) for minimal latency access. This Ethernet-based implementation circumvents PCIe-related bottlenecks and costs, allowing flexible CPU-to-accelerator combinations and scalability from single IPU-M2000 systems to extensive supercomputing setups comprising 64,000 IPUs, connected using standard networking at a lower cost compared to alternatives like InfiniBand [32].

IPU-Fabric represents a novel scale-out fabric tailored for machine intelligence communication, seamlessly integrated into IPU processors and IPU-M2000 systems. Distinguished by Compiled Communication and Bulk Synchronous Parallel protocol, IPU-Fabric ensures deterministic communication behavior. Each IPU features dedicated IPU-Links providing bidirectional bandwidth of 64 GB/s and aggregate bandwidth of 320 GB/s per chip. IPU-M2000 units incorporate eight external IPU-Links for intra-rack communication, facilitated by OSFP copper cables. The intra-rack configuration, termed IPU-POD16, encompasses

four IPU-M2000s interconnected in a daisy chain topology utilizing IPU-Links. Host–Link connectivity originates from the Gateway via PCIe NIC or SmartNIC card.

The memory model of the IPU machine is equally distinctive, with each IPU-M2000 system integrating DDR memory accessible to its constituent IPUs. Poplar Graph Compiler establishes deterministic code–memory relationships in both tile and DDR memory, allowing standalone utilization of this additional memory for inference processing without host server attachment. Moreover, the BSP model, compiling both computation and communication, minimizes network communication overhead compared to traditional messaging or shared memory constructs.

In conclusion, the incorporation of built-in fabrics is becoming imperative for AI accelerators, especially with the burgeoning sizes of models necessitating distribution across thousands of processors for timely processing. Graphcore's hybrid model, featuring proprietary IPU-Link fabric for intra-tile and intra-rack communication, complemented by tunneling IPU-Link protocol across standard 100GbE for rack-to-rack scale-out, supports larger configurations [32]. This disaggregated scaling model, coupled with IPU-Fabric, facilitates flexible configuration of multiple accelerators, enhancing versatility in AI computing scenarios.

4. Selected Graph Algorithms

To analyze the performance of the BSP model on the Graphcore IPU, three graph algorithms were selected: PageRank, Single-Source Shortest Path (SSSP) and Breadth-First Search (BFS). These iterative algorithms can be neutrally adapted to the BSP model. Moreover, each is supported by common graph analysis libraries, such as cuGraph or Katana [25,33]. Together, these algorithms cover a broad spectrum of graph types, making them a formidable trio for addressing diverse graph representation challenges across domains such as information retrieval, network analysis and optimization problems. Their adaptability and efficiency make them indispensable tools in the realm of graph theory and data science. Their adaptability across various graph types, combined with their efficiency, positions these algorithms as indispensable tools in data science applications. Whether it is uncovering hidden relationships, optimizing resource allocation or navigating complex networks, this formidable trio provides a versatile and powerful solution to diverse graph representation challenges. Let us now describe them in some detail.

PageRank is a link analysis algorithm that was developed by Larry Page and Sergey Brin, the co-founders of Google, as part of their early work on the Google search engine. The algorithm is designed to rank web pages in search engine results, and it forms the foundation of Google's search algorithm [34]. The PageRank algorithm is used to "rank" the vertices of the graph when "compared" to other nodes. The entire formula, used by Google, has not been published. However, its general iterative form is as follows (see [34]):

$$PR_x = \frac{1-d}{N} + d\left(\frac{PR_y}{L_Y} + \frac{PR_x}{L_x} \cdots + \frac{PR_z}{L_z}\right)$$

Here, PR_x is the PageRank of the node `x`; `d` is the damping factor; `N` is the number of nodes; L_x is the degree of the node `x`. The algorithm can be described as finding a Markov chain for vertices in a graph, or as a matrix diagonalization. The works in which the PageRank algorithm was optimized can be found, among others, in [26,35–41].

Works [35–37] describe optimization based on accelerating the convergence of PageRank values using eigenvectors. Hardware acceleration approaches have been also used, such as the application of 3D DRAM [26] or an FPGA implementation [42]. The 3D DRAM can reduce communication for the discussed algorithm, while the FPGA can be reprogrammed to create a dedicated circuit for the PageRank. Next, [38] examines the use of Map Reduce for PageRank. Various distributions of vertices, such as the adaptive method, i.e., a dynamic distribution with iterations [43], or one thread per one node distribution [39], have been tried. Use of the parallel Monte Carlo method in PageRank is explored in [40]. Finally, to speed up calculations, mixed precision was used in [41]. All this illustrates

the popularity and importance of this algorithm and its parallel realization. Algorithm 1 presents a generalized version of the PageRank algorithm.

Algorithm 1: PageRank Algorithm

Require: G (graph), d (damping factor), ϵ (convergence threshold)
  Initialize PageRank scores $PageRank[v]$ to $1/|V|$ for each node v
  repeat
    convergence $\leftarrow$ true
    for each node v in G **do**
      newPageRank $\leftarrow (1-d)/|V|$
      for each node u connected to v **do**
        newPageRank $\leftarrow$ newPageRank $+d \times (PageRank[u]/\text{outDegree}(u))$
      end for
      if $|\text{newPageRank} - \text{PageRank}[v]| > \epsilon$ **then**
        convergence $\leftarrow$ false
      end if
      PageRank[v] $\leftarrow$ newPageRank
    end for
  until convergence
  return PageRank

The second algorithm that has been experimented with is the SSSP. The problem is to find the shortest path between a node and the remaining nodes in the directed weighted graph. This algorithm can be used, for example, to determine the shortest travel route by public transportation. Popular solutions to the problem are the Bellman–Ford and the Dijkstra's algorithms. However, the Bellman–Ford algorithm was chosen for its better parallelization potential. Its pseudocode is summarized in Algorithm 2.

Algorithm 2: Bellman–Ford Algorithm for Shortest Paths

Require: G (graph), $start$ (source vertex)
  Initialize $distance[v]$ to ∞ for each node v
  $distance[start] := 0$
  for i from 1 to $|V|-1$ **do**
    for each edge from x to y **do**
      if $distance[x] > distance[y] + weight[x][y]$ **then**
        $distance[x] := distance[y] + weight[x][y]$
      end if
    end for
  end for

In each iteration, the algorithm checks each edge, whether it will have a shorter path when passing through it. This method is called the relaxation method, and it consists of checking whether a given vertex cannot obtain a shorter path for two other specific vertices. The algorithm is widely studied, and its optimized realizations have been discussed in [13,44,45].

The last considered algorithm is Breadth-First Search (BFS). The BFS algorithm is a fundamental graph traversal algorithm that explores a graph "level by level", visiting all the neighbors of a node before moving on to the next level. It is commonly used to find the shortest path in an unweighted graph and is also a key component in many other graph algorithms. BFS finds a path to all nodes in directed (or undirected) unweighted graphs. The BFS algorithm can be used to find all connected nodes in a graph or to check if the graph is bipartite. In the search, the first-in-first-out queue structure is used to traverse all nodes. Recent modifications of the algorithm have been reported in [46–48]. Algorithm 3 outlines how BFS works.

Algorithm 3: Breadth-First Search (BFS) Algorithm

```
Require: G (graph), start_node (source node)
  Create an empty set visited to keep track of visited nodes
  Create an empty queue and enqueue start_node
  Add start_node to the set visited
  while the queue is not empty do
    current_node ← dequeue from the front of the queue
    Process current_node (e.g., print it)
    for each neighbor neighbor of current_node do
      if neighbor is not in the set visited then
        Mark neighbor as visited
        Enqueue neighbor to the queue
        Add neighbor to the set visited
      end if
    end for
  end while
```

The BFS algorithm can use heterogeneous hardware acceleration approaches, i.e., use of multiple architectures in one system [46]. In the case of parallelization, data partitioning approaches have been investigated [47]. In some methods, virtual vertices have been added for better partitioning [49]. As with PageRank, adaptive strategies have been used [48]. In [29], the BFS algorithm was implemented in the BSP model on the Graphcore IPU processor and was shown to be more efficient than the Nvidia V100-based realization. Note that when dealing with very large graphs, special care is required to realize BFS on distributed memory systems. For instance, an approach to reduce the number of reads and writes to a disk is described in [50]. As with the SSSP, preprocessing is tested to speed up the BFS algorithm itself [51].

Approaches described in Section 2, and in the above discussion, show the breadth of issues faced in graph algorithms parallelization. However, it is also very clear that more work is needed to evaluate the actual application of the BSP model. This research gap is addressed in what follows.

5. Experimental Results and Verification

Overall, experiments have been focused on establishing base performance of two generations of the Graphcore IPU system running the BSP model-based implementations of three different graph algorithms. Moreover, comparison with CPU, and GPU based systems, using optimized libraries, provided by their manufacturers, was completed. The description of tested architectures is provided in Table 1. It should be mentioned that the Graphcore Bow system is basically the same system as the Graphcore MK2; the only difference is a new version (higher clock speed) of the IPU, and the rest of the configuration remains unchanged.

The Intel Katana library was used on the CPU processor. It contains the optimized versions of all three considered algorithms [25]. For the GPU (dubbed A-100), the cuGraph (version 22.4.0, [33]) library was used. It is a part of the RAPIDS library package, incubated by Nvidia. For the Graphcore IPUs (dubbed MK2 and Bow), the algorithms were implemented in C++ using the Poplar and PopLibs libraries provided by Graphcore. Each algorithm was formulated in terms of the BSP model and followed the approach found in [29].

For the testing purposes, several synthetic data sets were generated, and in all cases, each device/algorithm was tested on the same data. They were created by varying two parameters: (1) the number of nodes and (2) the edge factor, i.e., the average number of edges per node. The data generated had a maximum of one million edges. This was caused by the per tile memory limitation of the IPU processor (up to 624 KB). Additionally, multiple graph sizes had been evaluated. The data sets were selected in such a way to be

able to test denser and sparser graphs. Specifically, edge factors 20, 40 and 80 have been tested. However, due to space limitation, the results obtained for the largest edge factor (80) are mainly reported. However, the results obtained for the other edge factors are similar and confirm the reported findings.

Table 1. Configurations of all tested platforms.

	Single Socket Intel Xeon Gold 6138	**CPU+GPU Nvidia A-100**	**Dual CPU+4x Graphcore MK2**	**Dual CPU+4x Graphcore Bow**
Chip speed (Mhz)	2000	765	1325	1700
Cores number	20	6912	1472	1472
L1 cache	32 KB	192 KB	NA	NA
L2 cache	256 KB	40,960 KB	NA	NA
L3 cache	27.5 MB	NA	NA	NA
RAM	16 GB	40 GB	900 MB	900 MB
OS version	Ubuntu 18.04.4 LTS	Ubuntu 18.04.4 LTS	Ubuntu 18.04.4 LTS	Ubuntu 18.04.4 LTS
C++ Compiler	Clang 6.0.0-1	CUDA 11.7	Poplar SDK 2.4	Poplar SDK 2.4

First, in Figure 4, PageRank performance results are reported. Overall, the advantage of the IPUs is clear. Moreover, as expected, the Bow outperforms the MK2. The advantage of the IPUs over the A-100 is decreasing, as the data size increases. The CPU does well for the smallest graphs, outperforming both the A-100 and the MK2. However, for large graphs, it is by far the weakest. Up to 100,000 edges, the IPU performs almost perfectly without any real degradation in the observed execution time, and then we observe a stabilized but nevertheless slow increase in the time required to process an increasingly large graph. Nevertheless, even for 1,000,000 edges, MK2 and BOW perform much better than A-100.

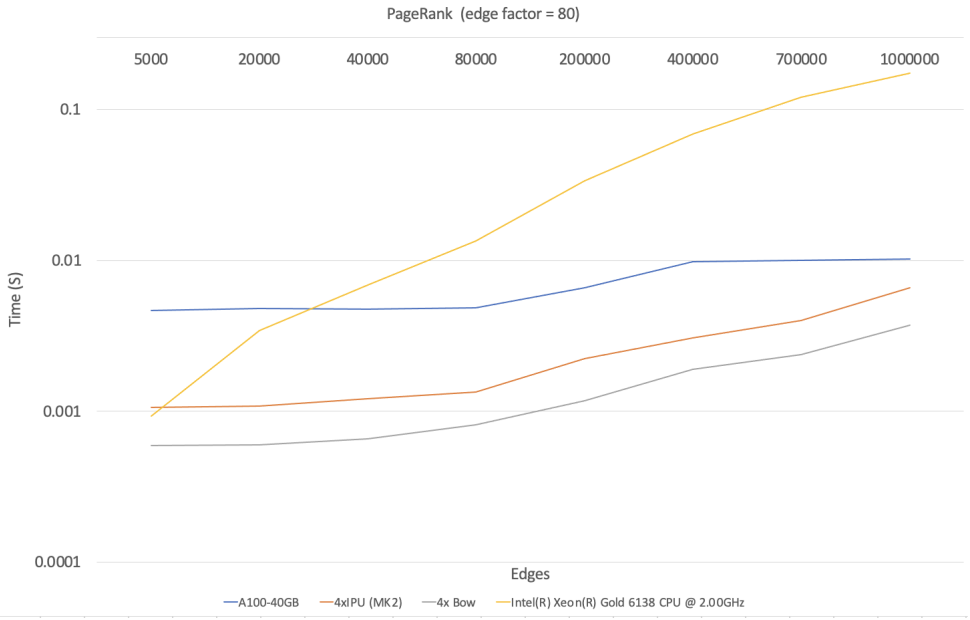


Figure 4. PageRank execution time for edge factor = 80.

The results for SSSP are presented in Figure 5. They show the clear advantage of the IPUs over the A-100 and the CPU. The A-100, MK2 and Bow achieve similar results for most data sets, with a slight increase in the compute time for the largest one. Initially, the CPU outperforms the A-100, but it slows dramatically for the largest graphs. In all cases, the Bow outperforms the MK2.

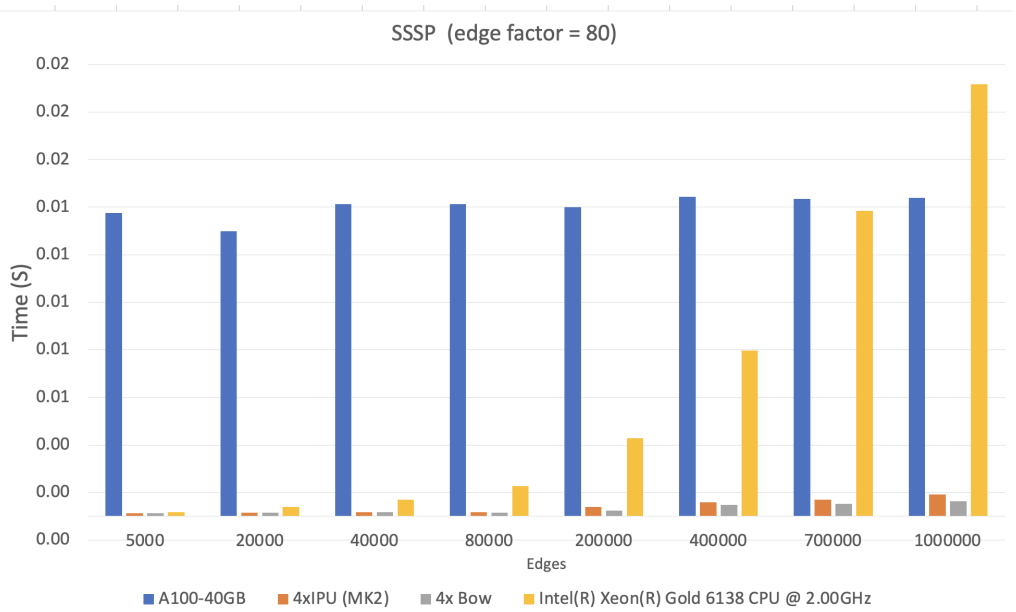


Figure 5. SSSP execution time for egde factor = 80.

For the BFS algorithm, results are presented in Figure 6. Here, the substantial advantage of both IPUs over A-100 and CPU remains. Again, initially, the CPU is faster than the A-100. However, it becomes approximately 30 times slower for the largest graphs. The Bow and the MK2 achieve similar performance. Both IPUs outperform the A-100 and the CPU, being up to one hundred times faster.

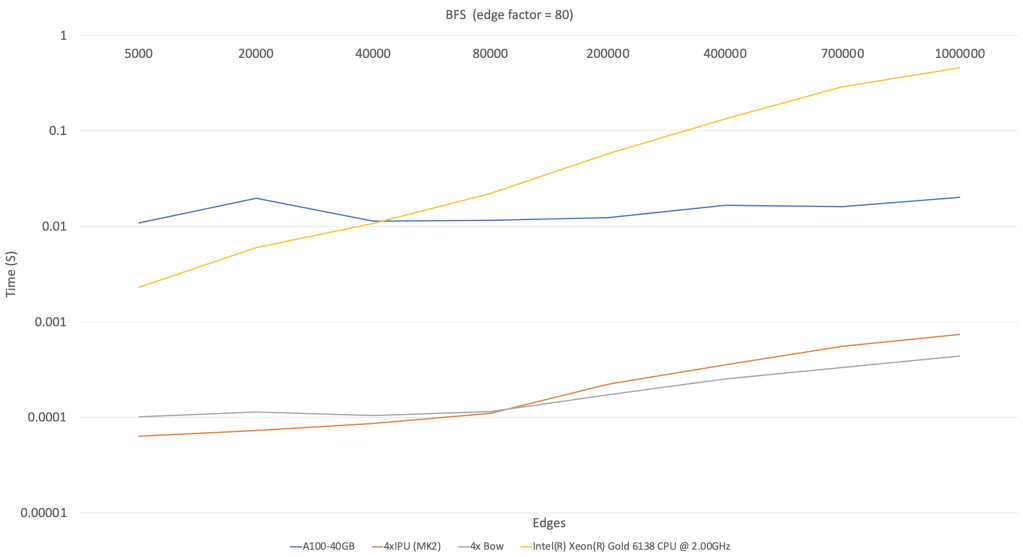


Figure 6. BFS execution time for egde factor = 80.

One of key the aspects of high-performance computing is the communication between components. The data load time depends not only on the device but also on the cluster configuration. The faster the data reach the computing unit, the shorter the waiting time for the result. Therefore, load time measurements were performed for all devices. Since the density of the graph does not matter here, experiments were run for the edge factor of 20 and have been depicted in Figure 7.

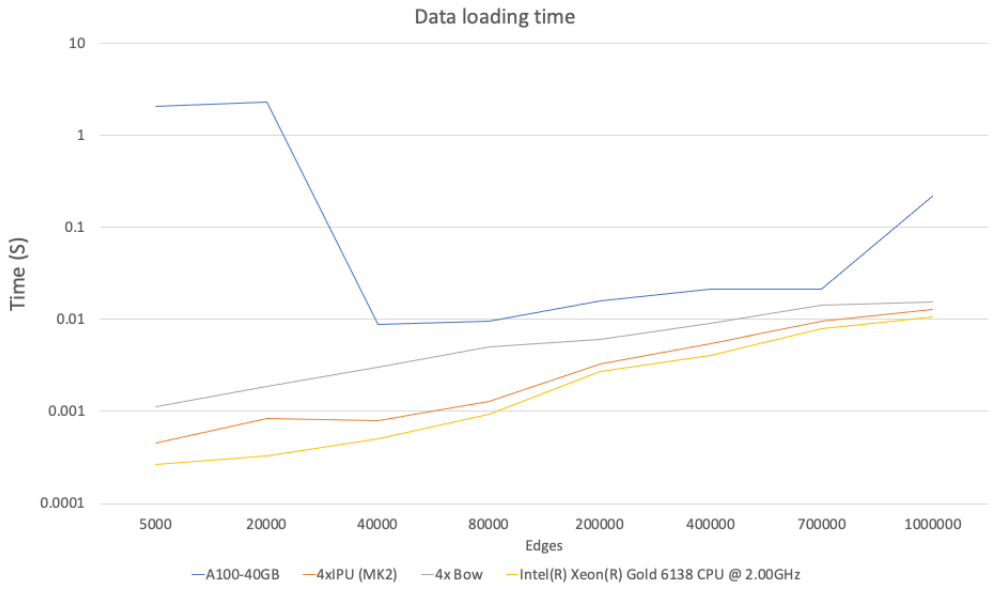


Figure 7. Loading time measurements for previously used data sets.

As expected, the CPU loaded the data the fastest. The next device to load data the fastest was the MK2, followed by the Bow. The loading times for the MK2, the Bow and the CPU converged with increasing data size. The A-100 had the longest load time, where the time was extremely long for the smallest and largest data sets. For medium data sets, it was similar to the Bow.

The last aspect that has been analyzed was the impact of the graph density on the performance. Only performance of Bow is reported, as the differences between the Bow and MK2 were negligible and constant. The tests were performed for all three algorithms. As expected, in all cases, processing efficiency increases with graph size and density. For a smaller number of edges, graphs with different densities were processed with the same performance. For the larger graph sizes, the performance for the densest graphs was up to two times higher. Moreover, the results show an increase in performance when a larger number of IPUs is used. A particular increase in efficiency was found for the largest graphs. When comparing performance of four and eight IPUs for BFS, the performance increase was at the level of ×1.6 and even ×2.19 for graphs with over 200,000 edges (i.e., superlinear scaling has been observed). When comparing between two and four IPUs, the scalability is ×2.9, which is also superlinear. This is a well-known effect related to the number of vertices stored per tile. When a very dense graph is distributed among a larger number of processing units, the overall system performance improves (see [52] for remarks on similar behavior observed in the context of solving the 3D Stokes equation on parallel computers).

6. New Insights and Contributions

This research delves into the effectiveness of Graphcore's IPUs for running common graph algorithms—Breadth-First Search (BFS), Single-Source Shortest Path (SSSP) and PageRank—using the Bulk Synchronous Parallel (BSP) model. It sheds light on several key areas that address current knowledge gaps:

- IPU Advantage for Specific Tasks: This study reveals a clear performance benefit for Graphcore IPUs (both MK2 and Bow generations) compared to A-100 GPUs and CPUs. This advantage is particularly pronounced for BFS and SSSP algorithms applied to large graphs with hundreds of thousands of edges. While PageRank also shows improvement on IPUs, the performance gap between A-100 and IPU narrows with increasing data size.
- Cross-Platform Performance Comparison: This work offers a valuable comparison of execution times for the three graph algorithms across CPUs, A-100 GPUs and

two generations of Graphcore IPUs (MK2 and Bow). This side-by-side analysis highlights the strengths and weaknesses of each platform for tackling graph processing tasks.
- Impact of Graph Density: This research investigates how graph density influences performance. The findings demonstrate that processing efficiency increases for all algorithms as graphs become denser. This is particularly evident for the densest graphs on larger graph sizes.
- IPU Scalability Potential: This study explores the scalability of IPU systems by comparing performance with different numbers of IPUs used for BFS. It reveals a superlinear performance increase (up to 2.5×) when doubling the number of IPUs. This suggests efficient processing of graphs with millions of edges on larger configurations.
- Efficient Data Loading: The experiments confirm the effectiveness of data loading capabilities in the IPU machine. Load times increase linearly, with no significant issues observed related to data size. This indicates a well-configured system for data transfer.

Building on the Momentum: This research aligns with recent studies showcasing the remarkable performance of parallel graph algorithms on Graphcore IPUs [53–56]. These studies explore various aspects, including hardware–software co-design and innovative algorithm optimizations for specific graph operations. Collectively, they illuminate the powerful capabilities of Graphcore's IPU architecture for efficient execution and acceleration of parallel graph algorithms. This work complements these findings by providing a comparative analysis across different hardware platforms and exploring the impact of graph density on performance.

In essence, this research addresses the need for a deeper understanding of Graphcore IPUs' potential for graph analytics. By offering a comparative analysis, investigating the impact of graph density and exploring scalability, this work provides valuable insights for researchers and developers working on parallel graph processing solutions. The observed performance improvements for BFS, SSSP and PageRank algorithms highlight the promise of IPU technology for accelerating real-world graph applications.

7. Concluding Remarks

Performance results and hardware comparisons show the excellent performance of BSP-based approaches and IPU processors for the explored graph algorithms. Particularly high advantage of Bow and MK2 was noted for the Single Source Shortest Path and the Breadth-First Search algorithms. For the PageRank algorithm, the advantage over an A-100 GPU decreased for the largest graphs. Another conclusion is the confirmation of the poor performance of the CPU when processing graphs with hundreds of thousands of edges. The CPU was able to achieve competitive performance only on the smallest graphs. In addition, the technological progress of the Bow over the MK2 processor is clear, with a constant increase in performance for the SSSP and the PageRank algorithms.

The data load time tests show the high quality of the IPU machine configuration, as the load time increased linearly and no problem-size-related deficiencies were noted. In the case of the A-100, significant load time deviations have been observed. This could indicate problems with the configuration of the particular cluster used for the experiments, causing data transfer delays. However, this observation is out of the scope of this contribution.

It was also observed that an IPU processor works best with dense graphs. This is due to the nature of the processor, which works using the BSP programming model and cannot communicate during calculations. Here, for obvious reasons, a lot more communication and synchronization is required for sparse graphs. This observation was further confirmed by analyzing the program execution graph generated by the compiler.

The last observation concerns scalability of the IPU. For BFS, superlinear performance increase (up to 2.5×) was observed when doubling the number of processors. This suggests that for configurations such as POD 256 (system consisting of 256 IPUs), efficient processing of graphs with millions of edges can be expected.

The achieved results show that the Graphcore IPU processor and the Bulk Synchronous Parallel technique have substantial potential for graph processing. The IPU can be used for the above-mentioned real-world use cases, significantly reducing the algorithm response waiting time and speed-up. The results demonstrate also the potential of the IPU for other graph algorithms. Hence, further results will be reported in subsequent publications.

Author Contributions: Conceptualization, P.G., M.P. and L.M.; Methodology, M.G.; Software, P.G. and B.K.; Validation, M.P.; Formal analysis, B.K. and M.G.; Investigation, B.K. and T.O.; Data curation, B.K.; Writing—original draft, P.G.; Writing—review & editing, M.P., M.G. and L.M.; Project administration, T.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: This paper and the research behind it would not have been possible without the exceptional support of Graphcore Customer Engineering and Software Engineering team. We would like to express our very great appreciation to Hubert Chrzaniuk, Krzysztof Góreczny and Grzegorz Andrejczuk for their valuable and constructive suggestions connected to testing our algorithms and developing this research work. This research was partly supported by PLGrid Infrastructure at ACK Cyfronet AGH, Krakow, Poland.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Schulz, F.; Wagner, D.; Zaroliagis, C. Using Multi-level Graphs for Timetable Information in Railway Systems. In Proceedings of the Algorithm Engineering and Experiments, San Francicsco, CA, USA, 4–5 January 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 43–59.
2. Fan, W.; Ma, Y.; Li, Q.; He, Y.; Zhao, E.; Tang, J.; Yin, D. Graph Neural Networks for Social Recommendation. In Proceedings of the Association for Computing Machinery, Atlanta, GA, USA, 29–31 January 2019; pp. 417–426. [CrossRef]
3. Michael, G.; Rolf, J.; Ypm, F.; Romero-Garcia, R.; Price, S.; Suckling, J. Graph theory analysis of complex brain networks: New concepts in brain mapping applied to neurosurgery. *J. Neurosurg. JNS Am. Assoc. Neurol.* **2016**, *124*, 1665–1678. [CrossRef] [PubMed]
4. Li, X.; Liu, S.; Li, Z.; Han, X.; Shi, C.; Hooi, B.; Huang, H.; Cheng, X. FlowScope: Spotting Money Laundering Based on Graphs. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; pp. 4731–4738. [CrossRef]
5. Henderson, R. Using graph databases to detect financial fraud. *Comput. Fraud. Secur.* **2020**, *2020*, 6–10. [CrossRef]
6. Zhang, D.; Liu, Z.; Jia, W.; Liu, H.; Tan, J. Path Enhanced Bidirectional Graph Attention Network for Quality Prediction in Multistage Manufacturing Process. *IEEE Trans. Ind. Inform.* **2020**, *18*, 1018–1027. [CrossRef]
7. Suzumura, T.; Zhou, Y.; Barcardo, N.; Ye, G.; Houck, K.; Kawahara, R.; Anwar, A.; Stavarache, L.; Klyashtorny, D.; Ludwig, H.; et al. Towards Federated Graph Learning for Collaborative Financial Crimes Detection. *arXiv* **2019**, arXiv:1909.12946.
8. Robinson, D.; Scogings, C. The detection of criminal groups in real-world fused data: Using the graph-mining algorithm, "GraphExtract". *Secur. Inform.* **2018**, *7*, 2. [CrossRef]
9. Fensel, A.; Akbar, Z.; Kärle, E.; Blank, C.; Pixner, P.; Gruber, A. Knowledge Graphs for Online Marketing and Sales of Touristic Services. *Information* **2020**, *11*, 253. [CrossRef]
10. Gepner, P. Machine Learning and High-Performance Computing Hybrid Systems, a New Way of Performance Acceleration in Engineering and Scientific Applications. In Proceedings of the 16th Conference on Computer Science and Intelligence Systems, Online, 2–5 September 2021; pp. 27–36. [CrossRef]
11. Superclouds: AI, Cloud-Native Supercomputers Sail into the TOP500. Available online: https://blogs.nvidia.com/blog/2021/06/28/top500-ai-cloud-native/ (accessed on 1 January 2024).
12. Hu, L.; Zou, L.; Liu, Y. Accelerating triangle counting on GPU. In Proceedings of the 2021 International Conference on Management of Data, Virtual, 18–22 June 2021; pp. 736–748.
13. Harish, P.; Narayanan, P. Accelerating large graph algorithms on the GPU using CUDA. In Proceedings of the International Conference on High-Performance Computing, Goa, India, 18–21 December 2007; pp. 197–208.
14. Lü, Y.; Guo, H.; Huang, L.; Yu, Q.; Shen, L.; Xiao, N.; Wang, Z. GraphPEG: Accelerating graph processing on GPUs. *Acm Trans. Archit. Code Optim. (TACO)* **2021**, *18*, 1–24. [CrossRef]
15. Song, L.; Zhuo, Y.; Qian, X.; Li, H.; Chen, Y. GraphR: Accelerating graph processing using ReRAM. In Proceedings of the 2018 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Vienna, Austria, 24–28 February 2018; pp. 531–543.

16. Zhong, J.; He, B. Towards GPU-accelerated large-scale graph processing in the cloud. In Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK, 2–5 December 2013; pp. 9–16.
17. Betkaoui, B.; Thomas, D.; Luk, W.; Przulj, N. A framework for FPGA acceleration of large graph problems: Graphlet counting case study. In Proceedings of the 2011 International Conference on Field-Programmable Technology, New Delhi, India, 12–14 December 2011; pp. 1–8.
18. Zhou, S.; Kannan, R.; Zeng, H.; Prasanna, V. An FPGA framework for edge-centric graph processing. In Proceedings of the 15th ACM International Conference on Computing Frontier, Ischia, Italy, 8–10 May 2018; pp. 69–77.
19. Khoram, S.; Zhang, J.; Strange, M.; Li, J. Accelerating graph analytics by co-optimizing storage and access on an FPGA-HMC platform. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 239–248.
20. Zeng, H.; Prasanna, V. GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms. In Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, 23–25 February 2020; pp. 255–265.
21. Wang, Y.; Hoe, J.; Nurvitadhi, E. Processor assisted worklist scheduling for FPGA accelerated graph processing on a shared-memory platform. In Proceedings of the 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, 28 April–1 May 2019; pp. 136–144.
22. Ma, X.; Zhang, D.; Chiou, D. FPGA-accelerated transactional execution of graph workloads. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 227–236.
23. Penders, A. Accelerating Graph Analysis with Heterogeneous Systems. Master's Thesis, University of Twente, Enschede, The Netherlands, 2012.
24. Zhou, S.; Prasanna, V. Accelerating graph analytics on CPU-FPGA heterogeneous platform. In Proceedings of the 29th International Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD), Campinas, Brazil, 17–20 October 2017; pp. 137–144.
25. Intel. Katana's High-Performance Graph Analytics Library. 2021. Available online: https://www.intel.com/content/www/us/en/developer/articles/technical/katana-high-performance-graph-analytics-library.html (accessed on 1 January 2024).
26. Sadi, F.; Sweeney, J.; McMillan, S.; Hoe, J.; Pileggi, L.; Franchetti, F. Pagerank acceleration for large graphs with scalable hardware and two-step spmv. In Proceedings of the 2018 IEEE High Performance extreme Computing Conference (HPEC), Waltham, MA, USA, 25–27 September 2018; pp. 1–7.
27. Angizi, S.; Sun, J.; Zhang, W.; Fan, D. Design, Automation & Test in Europe Conference & Exhibition (DATE). In Proceedings of the GraphS: A Graph Processing Accelerator Leveraging SOT-MRAM, Florence, Italy, 25–29 March 2019. [CrossRef]
28. Kapre, N. Custom FPGA-based soft-processors for sparse graph acceleration. In Proceedings of the 2015 IEEE 26th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Toronto, ON, Canada, 27–29 July 2015. [CrossRef]
29. Burchard, L.; Moe, J.; Schroeder, D.; Pogorelov, K.; Langguth, J. iPUG: Accelerating breadth-first graph traversals using manycore Graphcore IPUs. In Proceedings of the International Conference on High Performance Computing, Barcelona, Spain, 10–14 December 2021; pp. 291–309.
30. Caraballo-Vega, J.; Smith, N.; Carroll, M.; Carriere, L.; Jasen, J.; Le, M.; Li, J.; Peck, K.; Strong, S.; Tamkin, G.; et al. Remote Sensing Powered Containers for Big Data and AI/ML Analysis: Accelerating Science, Standardizing Operations. In Proceedings of the 2022 IEEE International Geoscience and Remote Sensing Symposium, Kuala Lumpur, Malaysia, 17–22 July 2022; pp. 4034–4037. [CrossRef]
31. Jia, Z.; Tillman, B.; Maggioni, M.; Scarpazza, D. Dissecting the Graphcore IPU Architecture via Microbenchmarking. *arXiv* **2019**, arXiv:1912.03413.
32. Freund, K.; Moorhead, P. The Graphcore Second-Generation IPU. Moor Insights & Strategy. 2020. Available online: https://www.graphcore.ai/hubfs/MK2-%20The%20Graphcore%202nd%20Generation%20IPU%20Final%20v7.14.2020.pdf?hsLang=en (accessed on 1 January 2024).
33. cuGraph GPU Graph Analytics. Available online: https://github.com/rapidsai/cugraph (accessed on 1 January 2024).
34. Langville, A.; Meyer, C. *Google's PageRank and Beyond: The Science of Search Engine Rankings*; Princeton University Press: Princeton, NJ, USA, 2011. [CrossRef]
35. Brezinski, C.; Redivo-Zaglia, M. The PageRank vector: Properties, computation, approximation, and acceleration. *SIAM J. Matrix Anal. Appl.* **2006**, *28*, 551–575. [CrossRef]
36. Migallón, H.; Migallón, V.; Penadés, J. Non-Stationary Acceleration Strategies for PageRank Computing. *Mathematics* **2019**, *7*, 911. [CrossRef]
37. Nagasinghe, I. Computing Principal Eigenvectors of Large Web Graphs: Algorithms and Accelerations Related to Pagerank and Hits. Ph.D. Dissertation, Southern Methodist University, Dallas, TX, USA, 2010; pp. 1–114. Available online: https://eric.ed.gov/id=ED516370 (accessed on 1 January 2024).
38. Liu, C.; Li, Y. A Parallel PageRank Algorithm with Power Iteration Acceleration. *Int. J. Grid Distrib. Comput.* **2015**, *8*, 273–284. [CrossRef]

39. Migallón, H.; Migallón, V.; Penadés, J. Parallel two-stage algorithms for solving the PageRank problem. *Adv. Eng. Softw.* **2018**, *125*, 188–199. [CrossRef]
40. Avrachenkov, K.; Litvak, N.; Nemirovsky, D.; Osipova, N. Monte Carlo methods in PageRank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.* **2007**, *45*, 890–904. [CrossRef]
41. Grützmacher, T.; Cojean, T.; Flegar, G.; Anzt, H.; Quintana-Ortí, E. Acceleration of PageRank with Customized Precision Based on Mantissa Segmentation. *Assoc. Comput. Mach.* **2020**, *7*, 1–19. [CrossRef]
42. Mughrabi, A.; Ibrahim, M.; Byrd, G. QPR: Quantizing PageRank with Coherent Shared Memory Accelerators. In Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium, Portland, OR, USA, 17–21 May 2021; pp. 962–972. [CrossRef]
43. Rungsawang, A.; Manaskasemsak, B. Parallel adaptive technique for computing PageRank. In Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Montbeliard-Sochaux, France, 15–17 February 2006. [CrossRef]
44. Köhler, E.; Möhring, R.; Schilling, H. Acceleration of shortest path and constrained shortest path computation. In Proceedings of the International Workshop on Experimental and Efficient Algorithms, Santorini Island, Greece, 10–13 May 2005; pp. 126–138.
45. Wei, W.; Yang, W.; Yao, W.; Xu, H. Accelerating the shortest-path calculation using cut nodes for problem reduction and division. *Int. J. Geogr. Inf. Sci.* **2020**, *34*, 272–291. [CrossRef]
46. Daga, M.; Nutter, M.; Meswani, M. Efficient breadth-first search on a heterogeneous processor. In Proceedings of the 2014 IEEE International Conference on Big Data, Washington, DC, USA, 27–30 October 2014; pp. 373–382. [CrossRef]
47. Fu, Z.; Dasari, H.; Bebee, B.; Berzins, M.; Thompson, B. Parallel breadth first search on GPU clusters. In Proceedings of the 2014 IEEE International Conference on Big Data, Washington, DC, USA, 27–30 October 2014; pp. 110–118.
48. Merrill, D.; Garland, M.; Grimshaw, A. Scalable GPU graph traversal. *ACM SIGPLAN Not.* **2012**, *47*, 117–128. [CrossRef]
49. Wen, H.; Zhang, W. Improving Parallelism of Breadth First Search (BFS) Algorithm for Accelerated Performance on GPUs. In Proceedings of the 2019 IEEE High Performance Extreme Computing Conference, Waltham, MA, USA, 24–26 September 2019; pp. 1–7. [CrossRef]
50. Vastenhouw, B.; Bisseling, R. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Rev.* **2005**, *47*, 67–95. [CrossRef]
51. Jiang, Z.; Liu, T.; Zhang, S.; Guan, Z.; Yuan, M.; You, H. Fast and efficient parallel breadth-first search with power-law graph transformation. *arXiv* **2020**, arXiv:2012.10026.
52. Ganzha, M.; Georgiev, K.; Lirkov, I.; Paprzycki, M. An application of the partition method for solving 3D Stokes equation. *Comput. Math. Appl.* **2015**, *70*, 2762–2772. [CrossRef]
53. Bernard, F.; Zheng, Y.; Joubert, A.; Bhatia, S. High Performance Graph Analytics on Graphcore IPUs. In Proceedings of the 2021 IEEE International Conference on High Performance Computing (HiPC), Bengaluru, India, 17–20 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 30–41.
54. Jia, Z.; Han, S.; Emerling, A.; Qiao, X. Scalable Graph Algorithm Design and Optimization for Graphcore IPUs. In Proceedings of the 41st ACM SIGMOD International Conference on Management of Data, Portland, OR, USA, 12–17 June 2022; pp. 2645–2658.
55. Tang, Y.; Xu, Z.; Liu, Z.; Li, J. Accelerating Personalized Recommendation with Graph Neural Networks on Graphcore IPUs. In Proceedings of the 2023 International Conference on Information Technology and Computer Applications (ICITACEE), Semarang, Indonesia, 31 August–1 September 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–6.
56. Xu, L.; Luo, Z.; Li, H.; Chen, J. Scalable Training of Large Graph Neural Networks with Structural Attention on Graphcore IPUs. *arXiv* **2023**, arXiv:2003.03134.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.