Politecnico di Torino

Politecnico di Torino

Master Degree in Electronic Engineering

A.a. 2024/2025

Graduation Session October 2025

Subgraph Isomorphism Acceleration on HBM-based Data Center FPGAs using High-Level Synthesis

Supervisor:
Prof. Luciano Lavagno

Candidate:
Shahabuddin Danish

Abstract

Subgraph isomorphism is a fundamental NP-hard problem in graph theory and is important for applications like social network analysis and bioinformatics, and it is a significant computational challenge when processing very large and non-uniform datasets. While field-programmable gate arrays (FPGAs) provide energy-efficient platforms for creating specialized hardware to accelerate graph workloads, the performance of graph accelerators is often constrained by the memory subsystem's bandwidth. Worst-Case Optimal Join (WCOJ) algorithms have the property of bounding the size of intermediate results compared to traditional exploration-based methods and this shifts the primary performance bottleneck to memory access. This thesis addresses the memory bottleneck for this problem by studying the feasibility of architecturally expanding an existing low-power and high-performance WCOJ based subgraph isomorphism accelerator originally designed for embedded FPGAs with a 128-bit memory interface for deployment on modern data center FPGA platforms equipped with High Bandwidth Memory (HBM).

The motivation for this architectural redesign is justified by a preliminary benchmark study on memory subsystems of AMD Alveo™ platforms, specifically HBM2 on the Alveo™ U55C and DDR4 on the Alveo™ U250. The benchmark study showed that HBM provides an approximate 6.5x aggregate sequential bandwidth improvement (~382 GB/s) over traditional DDR4 (~59 GB/s), comparable read latency and significant reduction in write latency. These findings confirm that the original kernel's 128-bit interface would severely underutilize the target platform's capabilities and a wider datapath would be necessary for achieving maximum performance.

The primary contribution of this work is the comprehensive architectural redesign of the kernel's complete datapath to natively support a 512-bit physical memory bus to allow full utilization of the target platform's memory subsystem. This redesign is performed by maintaining the original graph data structures and using 128-bit logical instructions for graph primitives (vertices and edges) while packing four such instructions into a single 512-bit memory word. This packing/unpacking methodology is developed and applied to the complete datapath of the accelerator, from host-side data preparation to on-chip processing modules, and it required

significant modifications to the kernel's two major phases, preprocessing and the multiway join pipelines. The preprocessing stage is redesigned to unpack logical 128-bit data graph instructions from incoming 512-bit words before sorting and scattering them into the final hash table structures and the pipelined multiway join is redesigned to consume these wider data structures.

The proposed implementation is a fully-pipelined 512-bit native WCOJ accelerator developed in C++ using Vitis™ High-Level Synthesis and optimizes data movement to utilize the full memory bandwidth of data center FPGA platforms by architecturally aligning the kernel's datapath with the physical memory interface. The thesis also demonstrates a reproducible methodology for migrating and optimizing other memory-bound HLS designs for high-performance computing environments.

Acknowledgements

I wish to express my sincere gratitude to the people who have supported me throughout the course of this thesis. First and foremost, I extend my thanks to my supervisor, Professor Luciano Lavagno, for providing me with the opportunity to pursue this research and for his continuous support, insightful feedback, and availability, all of which have been very valuable for the successful completion of this thesis.

I am especially grateful to my thesis mentor, Roberto Bosio for his dedicated supervision, constant availability, patience, and technical guidance. This work would not have been possible without his constant support and encouragement.

Finally, on a personal note I would like to thank my fiancée, my parents, my entire family, and my friends for their collective love, support, encouragement, and prayers. Their belief in me has been a constant source of strength and motivation throughout the course of this work.

Table of Contents

List of Tables

List of Figures

List of Acronyms

BRAM
: Block Random-Access Memory

CLB
: Configurable Logic Block

CU
: Compute Unit

DDR
: Double Data Rate (Synchronous Dynamic RAM)

FIFO
: First In First Out

FPGA
: Field-Programmable Gate Array

HBM
: High Bandwidth Memory

HLS
: High-Level Synthesis

URAM
: Ultra Random-Access Memory

WCOJ
: Worst-Case Optimal Join

Chapter 1

Introduction

1.1 Graph Processing

A graph, in basic terms, is a collection of vertices and edges which connect them. Graphs are capable of modeling a large variety of complex real-world systems, and in the context of computing they provide a powerful and useful model for representing complex relationships and interactions between data and have become important for a lot of domains like mapping interactions in digital social networks, modeling protein-to-protein interactions in bioinformatics, detecting fraudulent transactions in financial systems, and optimizing logistics and supply chains.

The recent growth of digital services and widespread IoT sensor networks has created a huge growth in the size of graph datasets, where modern graphs that represent objects like the World Wide Web or large social media platforms can now consist of billions of vertices and trillions of edges, and the size and structural complexity of this data has created a challenge for traditional computing architectures [1].

The structure of real-world graphs is usually highly irregular, sparse, and follows a power-law distribution as compared to the dense, regular dense data structures in domains like linear algebra or scientific computing. This inherent irregularity creates random memory access patterns that fundamentally reduce performance of conventional CPU and GPU architectures, which include deep cache hierarchies and prefetching mechanisms and they rely on the properties of spatial and temporal locality of data to hide memory latency. These strategies mostly become ineffective when processing graph data structures and as a consequence, graph processing algorithms are usually memory-bound as they spend a large majority of execution time to fetch data from the main memory rather than performing computation. This bottleneck is usually referred to as "memory wall" in computing and it is the primary bottleneck for achieving high-performance graph processing. Therefore,

the development of specialized and energy-efficient hardware solutions which are capable of reducing this memory bottleneck is an important and active area of research in high-performance computing.

1.2 Subgraph Isomorphism Problem

Subgraph isomorphism is a fundamental and computationally intensive problem among different graph processing problems. Formally, given a small query graph $Q = (V_q, E_q, L_q)$ and a large data graph $G = (V_g, E_g, L_g)$, the subgraph isomorphism problem consists of finding all injective mappings $f : V_q \to V_g$ such that for every vertex v $\in V_q$, the label $L_q(v)$ equals $L_g(f(v))$, and for every edge $(u, v) \in E_q$, a corresponding edge $(f(u), f(v))$ exists in E_g. All these mappings, also known as embeddings, represent an occurrence of the query pattern within the larger data graph.

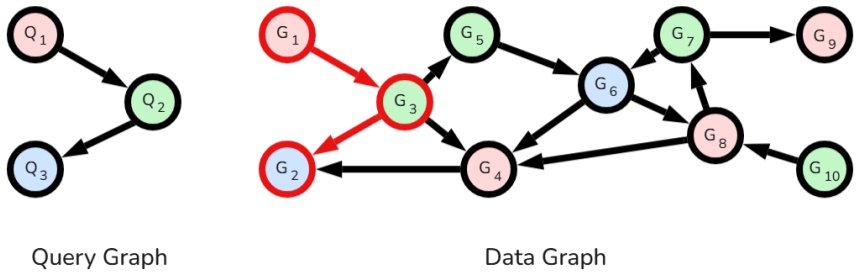

Figure 1.1: The highlighted subgraph $\{G_1, G_2, G_3\}$ within the Data Graph (right) represents a valid embedding of the Query Graph (left). The mapping preserves both the vertex labels (represented by colors) and the directed edge structure.

The highlighted subgraph in Figure 1.1 demonstrates a valid embedding by satisfying all constraints of the formal definition. The injective mapping f can be defined as:

- $f(Q_1) = G_1$
- $f(Q_2) = G_3$
- $f(Q_3) = G_2$

This mapping is verified based on the formal definition and it holds true against the verification of all constraints:

1. **Vertex Label Constraint:** $\forall\ v \in V_q, L_q(v) = L_g(f(v))$
   The label of each query vertex, represented by its color, matches the label of its corresponding data vertex. $L_q(Q_1)$ matches $L_g(G_1)$ (pink), $L_q(Q_2)$ matches $L_g(G_3)$ (green), and $L_q(Q_3)$ matches $L_g(G_2)$ (blue).

2. **Edge Structure Constraint:** $\forall\ (u, v) \in E_g,\ (f(u),\ f(v)) \in E_g$
   All directed edges in the query graph are preserved in the mapping. The edge $(Q_1,\ Q_2) \in E_q$ correctly maps to the edge $(f(Q_1),\ f(Q_2)) = (G_1,\ G_3)$ which exists in E_g, and similarly, the edge $(Q_3,\ Q_2) \in E_q$ maps to the existing edge $(f(Q_3),\ f(Q_2)) = (G_2,\ G_3)$ in the data graph.
3. **Injective Mapping:** The function is injective, as each of the three distinct query vertices Q_1, Q_2, and Q_3 maps to a unique vertex $(G_1,\ G_3,\ G_2)$ in the data graph.

Since all constraints are satisfied, the subgraph induced by the vertex set $\{G_1,\ G_2,\ G_3\}$ is a valid isomorphic instance of the query graph.

This computational problem is classified as **NP-hard**, which means that there is no known algorithm that can find a solution in polynomial time with respect to the input size, and this is a direct consequence of the combinatorial explosion of the potential solution space. A brute-force approach which would test all possible injective mappings from query vertices to data vertices becomes computationally unmanageable for graphs of even moderate size, and the number of mappings grows factorially, making the runtime of an exhaustive search to increase at a rate that is much larger than polynomial complexity.

The exponential growth characteristic becomes the primary barrier to scalable analysis because, although an algorithm can perform acceptably on a data graph with thousands of vertices, its runtime can increase from seconds to hours or even years as the graph scales to millions or billions of vertices, and this property makes standard approaches impractical for real-world graph datasets which are targeted by this thesis. This challenge has driven the development of two primary families of algorithms [2]. The first family of algorithms is **exploration-based methods** that usually use a backtracking search to recursively extend partial solutions by mapping one query vertex at a time. These algorithms are effective, but they can generate a large number of intermediate results, and their performance is fundamentally limited by the bottlenecks of memory bandwidth and latency because of irregular memory access patterns which are inherent in traversing graph data structures. Therefore, the problem of scalable subgraph isomorphism requires the development of optimized algorithms to effectively prune the large search space combined with specialized hardware accelerators which are designed to efficiently execute core operations of these optimized methods [2].

The alternative approach is to reframe the problem from a relational database perspective, in which each edge in the query graph is treated as a relation and the vertices are treated as attributes, which transforms the subgraph isomorphism problem into a multi-way join operation over these relations. This formulation of the problem allows different algorithms to utilize an extensive amount of research in query optimization. The most notable algorithms in this space that are of relevance

are **Worst-Case Optimal Join (WCOJ)** algorithms which have the very useful theoretical property of bounding the size of intermediate join results. This property is necessary for managing the computational and memory footprint of the algorithm, and by effectively managing the exponential growth of intermediate results, WCOJ algorithms shift the primary performance bottleneck away from computation and completely onto memory bandwidth. This is possible because in these algorithms, the dominant cost of enumerating the data graph becomes the repeated scanning of relations from main memory, and consequently, the performance of a WCOJ-based accelerator is fundamentally coupled to the efficiency of its data movement and the throughput of its memory subsystem [2].

1.3 High-Bandwidth Graph Acceleration using Data Center FPGAs

To overcome the "memory wall" problem in graph processing discussed in the previous section, research has increasingly focused on specialized hardware solutions and reconfigurable computing architectures like Field Programmable Gate Arrays (FPGAs) have become an attractive platform which offer a third framework of computing in the form of domain-specific accelerators alongside traditional CPU and GPU computing architectures. An FPGA is an integrated circuit whose architecture consists of a reconfigurable fabric of logic blocks (CLBs), on-chip memories (BRAMs and URAMs), and programmable interconnects, and this structure allows for the creation of custom digital circuits designed precisely for the computational and data movement patterns of a target algorithm using different digital design flows. The primary advantages of this approach are massive fine-grained parallelism, the ability to design custom datapaths, and superior energy efficiency (performance per watt), all of which are achieved by removing the overhead of general-purpose instruction processing.

The flexible, programmable hardware structure of FPGAs is especially well suited to challenges of the WCOJ-based subgraph isomorphism algorithm. The algorithm's structure consists of a sequence of dependent stages (Propose, Intersect, Verify) and these stages naturally map to a **deeply pipelined dataflow architecture**, which is describable in HLS, and this model allows for high-throughput processing where new data can enter the pipeline on every clock cycle, allowing to maximize computational efficiency. More importantly, FPGAs provide a direct solution to the memory bottleneck problem as data center FPGAs are often equipped with high bandwidth memory technologies such as **High Bandwidth Memory (HBM)**, which provides much higher throughput than traditional DDR memory through a wider physical interface (typically 512-bits per HBM channel or more), and unlike traditional fixed computing architectures (CPUs and GPUs) with fixed data

bus widths, an FPGA allows for the design of custom memory controllers and datapaths that can precisely match the physical width of the attached memory subsystem, enabling a domain-specific accelerator to fetch significantly more data per memory transaction and maximize memory throughput. Furthermore, the on-chip BRAMs and URAMs are also very useful for implementing low-latency caches and high-throughput FIFOs which are used to buffer partial embeddings between the pipeline stages in WCOJ-based algorithms, effectively hiding the latency of off-chip memory accesses.

The baseline LESS accelerator upon which this thesis is built [3], successfully demonstrated the viability of implementing the WCOJ algorithm on an FPGA and its design effectively translated the relational join operations into a high-performance HLS dataflow. However, as the LESS kernel was designed for an embedded platform, its performance was constrained by a **128-bit memory interface**, and while it proved the algorithmic approach to be sound, when the design is deployed on a modern data center FPGA which provides a 512-bit physical memory bus, the kernel's datapath can only utilize a quarter of the available capacity. This underutilization of the platform's memory bandwidth, also the primary requirement of the algorithm, creates a critical performance gap by failing to utilize the 75% of the available bandwidth offered by platform hardware and creating an artificial bottleneck in the accelerator's datapath.

Therefore, the research presented in this thesis focuses on directly addressing this architectural performance gap by utilizing the reconfigurability of FPGAs and adapting the proven baseline design through a datapath redesign to fully saturate the **512-bit HBM** systems of modern data center FPGAs.

1.4 Thesis Motivation and Problem Statement

The previous sections have established that the performance of graph processing, especially for memory-bound algorithms like the WCOJ, is fundamentally limited by the throughput of the memory subsystem, and that FPGAs provide an architectural model which is capable of addressing this bottleneck through development of a domain specific accelerator with a custom data path design. The baseline "LESS" accelerator successfully demonstrated the viability of mapping the WCOJ algorithm to a pipelined HLS architecture, which was achieved through a new design that integrated a complete on-chip preprocessing phase and eliminated the need for an energy expensive host CPU to build the required data structures for graph processing. The architecture of the accelerator was based on a flexible two-level hash table and custom caching mechanisms to mitigate off-chip memory latency and the original research showed that this approach was viable and the accelerator delivers a good balance of performance and power consumption compared to state-of-the-art

high-performance solutions on both CPUs and GPUs. Experimental results from the study showed that the LESS accelerator has a good advantage in performance-per-watt for complex memory-bound graph operations [3], and this demonstrated efficiency validates the feasibility of WCOJ on FPGA methodology as a reliable and powerful foundation for graph processing accelerators. However, the development context for embedded platforms imposed a 128-bit memory interface, creating an architectural limitation that prevents its effective scaling on contemporary high-performance hardware.

The motivation for this thesis comes from the major technological gap between the high memory throughput capabilities of modern data center FPGAs and the baseline accelerator's architectural datapath limitation of 128-bit as it was designed for an embedded FPGA platform. High Bandwidth Memory (HBM) technology presents a good opportunity for improving the performance of memory-bound workloads, and a performance study conducted in this thesis, which is detailed in Chapter 4, was performed to quantify this opportunity. The results revealed that the 512-bit HBM interface provides an approximate 6.5x aggregate sequential bandwidth advantage over a traditional 512-bit DDR4 interface, providing the opportunity for substantial performance gains. This empirical evidence makes it clear that deploying the 128-bit baseline kernel on a data center accelerator with a 512-bit physical HBM interface would create a severe architectural bottleneck, leaving 75% of the available memory bandwidth unutilized and negating the primary advantage of the target hardware.

Therefore, this thesis addresses the problem of this architectural mismatch. The core problem statement of the thesis is the sub-optimal performance of a proven WCOJ graph accelerator on high-performance data center FPGAs due to the mismatch of its datapath with the wide-bus memory subsystem that leads to severe underutilization of the available memory bandwidth, and the objective of this work is to resolve this bottleneck through architectural redesign and implementation of the accelerator, with the goal to evolve the kernel to natively support a 512-bit datapath and enable it to fully saturate the available memory bandwidth on the target FPGA. This will be achieved through a packed-instruction methodology that maintains logical 128-bit data structures while maximizing the physical memory bus utilization, all using a High-Level Synthesis design flow, and the final implementation will validate this approach and provide a high-performance solution for subgraph isomorphism enumeration on data center platforms.

1.5 Key Contributions

This thesis presents the adaptation, redesigning, implementation, and evaluation of a high-performance subgraph isomorphism accelerator based on an embedded

FPGA platform for HBM equipped data center FPGAs. The main contributions of the work are as follows:

1. **Performance Characterization of Data Center FPGA Memory Systems:** A detailed benchmark study was performed on AMD ALveo platforms to quantitatively analyze and compare the aggregate bandwidth and average access latency of HBM2 and DDR4 memory subsystems. The analysis provides the experimental data which justifies the necessity of a wide datapath and parallel architecture for memory-bound algorithms, which can be implemented using HBM memory platforms.
2. **Architectural Migration and Data Path Widening**: The baseline 128-bit LESS accelerator was successfully adapted from an embedded FPGA platform to the Vitis data center design flow, and the memory interface of the accelerator was updated to support a 512-bit wide physical memory bus with the help of HLS compiler directives, allowing the accelerator to utilize the full physical memory bus and higher bandwidth of the target platform.
3. **Implementation and Validation**: A complete, functionally verified 512-bit subgraph isomorphism accelerator was implemented on an AMD Alveo U55C HBM-based data center FPGA, and an XRT-based host application and a linker configuration for the multi-bank HBM subsystem were developed as the main components of this implementation.
4. **Experimental Performance Analysis**: A detailed performance analysis was performed which compared the final 512-bit HBM accelerator to the 128-bit DDR4 baseline. The analysis shows a significant performance improvement due to the wider physical memory bus and higher throughput on the data center platform, and this research also explores the limitations of having a wider physical bus with a serial or less parallel architecture when processing large and complex graph datasets.

1.6 Thesis Structure

The research presented in this thesis is organized in the following manner:

- **Chapter 2** provides a review of the background concepts and related work, and it covers the theoretical foundations of subgraph isomorphism algorithms including exploration-based methods, join-based methods, and it discusses the architectural characteristics of High Bandwidth Memory.
- **Chapter 3** details the architecture of the baseline LESS accelerator and describes the two-phase execution model of the design, the main WCOJ-based multiway join pipeline, and the main data structures and hardware optimizations.

- **Chapter 4** presents the methodology and results of the detailed performance analysis of different data center FPGA memory subsystems, and it describes the bandwidth and latency microbenchmark kernels and provides a quantitative comparison between HBM and DDR4, establishing the motivation for the accelerator's adaptation and redesign.
- **Chapter 5** explains the implementation of the high-bandwidth accelerator and details the migration to the Vitis data center design flow, the methodology for data path widening, and the final experimental evaluation, such as functional verification and a detailed performance analysis.
- **Chapter 6** concludes the thesis by summarizing the key findings and contributions, and it outlines the promising directions for future research, including detailed datapath redesign, optimization and multi-kernel scaling.

Chapter 2

Background

The subgraph isomorphism problem is NP-hard so it doesn't have any universally optimal algorithm, this is why there are different families of algorithms with different computational models and performance properties. There are comprehensive surveys like the book by Sun and Luo, which suggests that these algorithms can be broadly categorize into three major types which include exploration-based backtracking, constraint programming and relational join-based approaches [2]. Although this thesis focuses on the last one, it is important to review all three in order to have a full picture of the problem space.

2.1 Exploration based Algorithms

The exploration-based framework is the classic and best studied model of subgraph isomorphism that organizes the problem in state-space search terms. These algorithms use a backtracking approach to recursively extend a partial match, one query vertex at a time, until a complete embedding has been discovered or a conflict is observed, and these types of algorithms have started with the original Ullmann algorithm which has later been optimized with many state-of-the-art algorithms including VF2, its successor VF2++, and RI [2].

The implementation of such exploration based algorithms can be broken down into a three phase process and is abstracted by the generic framework given by Sun and Luo [2].

2.1.1 Filtering and Pruning

The first step is to aggressively prune the search space and produce a candidate set $C(u)$ of each query vertex $u \in V_q$, where a data vertex $v \in V_g$ only lies in $C(u)$ in the case that it may be used in a valid mapping of u, and this approach is

usually achieved by a series of increasingly advanced filters. A baseline Label and Degree Filter (LDF) is used to make sure that a mapping is possible, in which case the label of v has to be equal to that of u and the degree of v must be at least equal to the degree of u. There are also more sophisticated techniques that exist like the Neighbor Label Frequency (NLF) filter which is used in algorithms like CFL and DP-iso, and this filter further prunes candidates by making sure that the neighborhood of v has as many vertices of the same label as the neighborhood of u.

2.1.2 Matching Order Selection

The sequence in which query vertices are mapped, called the mapping order or Query Vertex Order (QVO), has a major impact on the performance, because an effective ordering gives higher priority to more selective vertices (e.g., vertices with smaller sets of candidates or higher degrees), which can help the backtracking search to find failures earlier and therefore prune large branches of the search tree from the search space, and different heuristic techniques are used like static analysis of the query graph structure or dynamic ordering based on the state of intermediate results, etc., are used to find a more efficient Query Vertex Order for this purpose.

2.1.3 Enumeration

The enumeration phase starts after candidate sets and an optimum matching order is determined, and during enumeration the algorithm recursively traverses the search space, and for the current query vertex u, it computes a set of local candidates from $C(u)$ that maintain connectivity with the data vertices which are already part of the partial embedding.

On one hand, exploration-based methods are highly optimized, but on the other hand, their weakness is in their memory access patterns. This is because the recursive, depth-first traversal of the search space results in irregular or true random (pointer-chasing) memory accesses, which have bad spatial and temporal locality and this memory access pattern is not optimized for modern CPU and GPU architectures whose execution depends on deep cache hierarchies and prefetching mechanisms for memory access, leading to the situation called “memory wall” where the algorithm is constantly stalled waiting for memory.

2.2 Join-Based Algorithm

The other type of algorithms redefine the subgraph isomorphism problem from a graph traversal problem to a relational database query. In this model of solving for subgraph isomorphism, each edge (u, v) of the query graph Q is associated with a

relation $R(u, v)$ and these relations include all the edges in the data graph G which connect vertices with the correct label, and the vertices u and v are considered as attributes of the relation. This transformation changes the problem of locating all the embeddings to be the same as performing a multi-way join across all relations which are derived from the query graph.

This general approach can be further divided into two different subgraph isomorphism solving strategies:

2.2.1 Pair Wise Join

This approach is a conventional database technique in which relations are joined in a binary tree manner. Although it is good for most types of queries, pair-wise joins are highly inefficient for cyclic queries, which are typically common in graph structures, and this approach leads to the problem where an intermediate result of a pair-wise join may be asymptotically larger than the final result and can cause an large waste of computation and memory bandwidth.

2.2.2 Worst-Case Optimal Join (WCOJ)

In order to resolve the problem of intermediate result explosion, state-of-the-art join-based graph algorithms use WCOJ theory. The WCOJ algorithms (e.g., Generic-Join, Leapfrog Triejoin) depend on the AGM inequality bound and they are known to have a provable dependence on the largest possible output size of the query for their running time, and these algorithms work by generating sets "at-a-time" and iteratively compute the candidates for a new vertex by performing intersection operations on all relevant relations simultaneously.

WCOJ algorithms provide an effective solution to the main performance limitation of pair-wise joins by limiting the size of the intermediate results, but this computational efficiency has some constraints, such as the in this case the dominant operation then becomes repeated scanning of relations from the main memory and their intersection. Consequently, the performance of a WCOJ-based algorithm can no longer be limited by computational complexity, but rather it becomes directly memory bound, which means that the execution speed of the algorithm directly depends on the bandwidth available in memory. This characteristic makes the WCOJ-based algorithms a good choice for hardware acceleration on compute hardware that contains high-throughput memory systems.

2.3 Constraint Programming

A third approach also exists which defines the subgraph isomorphism problem as a Constraint Satisfaction Problem (CSP), in which, the query vertices are considered as variables, the set of data vertices form the domain for the variables so they provide the possible values for the variables, and query edges are defined as the structural constraints on any valid assignment, and by defining the problem like this, there are algorithms like Glasgow [2] which use advanced solvers that use backtracking and inference to find every valid assignment. However, while CSP is a powerful and general approach, in the specific case of subgraph isomorphism, specialized exploration-based or join-based algorithms usually demonstrate better performance than such methods.

2.4 High Bandwidth Memory (HBM)

The increase of computational density in FPGAs and GPUs has further increased the "memory wall" problem, where performance is not constrained by on-chip processing power but rather by the rate at which data can be transferred to and from external memory. Conventional Double Data rate (DDR) SDRAM, although developing, has inherent physical limitations in scaling its interface width due to pin count constraints, signal integrity considerations, and power consumption, and due to these factors, High Bandwidth Memory (HBM) has been developed as an architectural improvement in memory systems, a topic explored in depth by microbenchmark studies such as the work by Lu et al. [4], because it is specifically designed to provide orders of magnitude of improvement in memory bandwidth within a limited power budget.

2.4.1 HBM Architecture

HBM utilizes an alternative die-stacking process which is either 2.5D or 3D, which is different from the traditional planar, two-dimensional layout of DDR modules on a printed circuit board, and in HBM multiple DRAM dies are vertically stacked at a time and connected using through-silicon vias (TSVs), and this stack is then placed alongside the processor (FPGA or GPU) on a standard silicon interposer substrate. This physical layout allows for the primary benefits of HBM:

- **Ultra-Wide Interface:** The short dense connections on the interposer substrate in HBM provide a very wide memory interface. For example, a single HBM2 stack can offer up to 1024 bits, which is a great difference from the 64-bit wide channel that exists within a single DDR4 DIMM. This wide

bus is the main cause of the huge bandwidth advantage of HBM as it allows to transfer a lot more data per clock cycle [4].

- **Independent Channels and Pseudo-Channels:** The physical interface of an HBM2 stack is not a single bus but rather it is logically partitioned into four or eight separate channels of 128-bits each, and then all these channels are further subdivided into two completely independent 64-bit pseudo-channels. This fine-grained partitioning is an important architectural feature which enables parallel access of concurrent memory requests so that they can be serviced simultaneously with lesser interference, reducing bank conflicts and improving effective bandwidth, especially when the memory workload has random or semi-random access patterns.
- **Power Efficiency:** The electrical signaling paths are significantly reduced by physically reducing the distance between the memory dies and the processing logic, which allows reduced signaling voltages and allows the memory I/O driver power to be reduced, resulting in much more energy efficient (performance-per-watt) interfaces compared to the relatively longer signal distances for DDR memory interfaces.

2.4.2 Architectural Comparison with DDR

The architectural differences of HBM from traditional DDR4 memory subsystem highlight some important differences which should be considered. The main difference in an HBM2 system is the aggregate bandwidth, which is also demonstrated in the benchmark study conducted for this thesis (detailed in Chapter 4). An HBM2 memory subsystem with a 512-bit wide physical bus can deliver an aggregate sequential throughput of approximately 382 GB/s, whereas a multi-channel DDR4 system on a comparable platform is limited to around 59 GB/s. This is ∼6.5x increase in throughput which is also the primary motivation for targeting HBM for memory-bound workloads.

This high bandwidth is possible in HBM due to its highly parallel architecture rather than just clock speed, and HBM clock frequencies are typically lower than those of high-performance DDR4. Consequently, since HBM achieves higher throughput using parallelism, the absolute latency for a single, random memory access can be comparable to, or in some cases slightly lower than DDR4. This is a drawback of HBM memory subsystems, however the higher degree of parallelism from the pseudo-channel architecture allows HBM memories to service a much larger number of concurrent memory access requests, effectively hiding this latency and achieving superior system-level throughput.

The main trade-offs for HBM are capacity and cost as the complex manufacturing process that involves TSVs and silicon interposer substrates makes HBM more expensive per gigabyte. Furthermore, another drawback is that since the

HBM architecture is fixed and on-package, it means that its capacity is not user-expandable and it can typically offer a lower maximum capacity compared to computing systems with multiple DDR4 DIMMs.

2.4.3 Implications for Subgraph Isomorphism Acceleration

The work of this thesis is focused on a WCOJ subgraph isomorphism algorithm that is known to have a memory-bound problem, therefore the architectural characteristics of HBM memory subsystems are very important for this specific problem. The performance of the WCOJ algorithm is directly correlated to the rate at which candidate sets and edge data can be read from external memory, and HBM provides a huge bandwidth advantage which is a clear opportunity for an order-of-magnitude performance improvement if the accelerator's data path is able to utilize the wider physical bus and therefore an architectural adaptation to a 512-bit data path is necessary to utilize the maximum HBM bandwidth with the objective of designing a kernel that is capable of issuing memory requests at a sufficient rate to saturate the high-bandwidth interface.

Chapter 3

Baseline LESS Accelerator Architecture

The research presented in this thesis develops an established and high-performance subgraph isomorphism accelerator called LESS (Low-power Energy-efficient Subgraph Isomorphism). The LESS kernel uses a Worst-Case Optimal Join (WCOJ) algorithm that is entirely implemented on the FPGA using HLS, and this design choice shifts the main performance bottleneck to memory bandwidth. The LESS kernel's performance is used as the baseline for the work in this thesis and its architecture is described in this chapter, which was initially designed for a 128-bit memory interface on an embedded FPGA platform. The significance of the architectural redesign of the LESS accelerator to adapt it for high-bandwidth data center FPGAs requires an understanding of the kernel's two-phase execution model and key data structures, which is the main contribution of this thesis.

3.1 Architectural Overview

LESS kernel is designed to operate in two distinct and non-overlapping phases, both of which are executed entirely on-chip, where the first is a **Preprocessing Phase** and then the second is a **Multiway Join (Enumeration) Phase**. The decision to run both the preprocessing and the enumeration phase on the FPGA is an architectural design choice [3] where the host CPU is not required to do the initial graph data structuring to make the acceleration energy efficient, which is a common requirement in many other FPGA-based graph accelerators.

The whole kernel is designed and implemented as a streaming architecture where data is passed between modules through hls::stream FIFOs. The top-level kernel is interfaced with the host and off-chip memory through multiple AXI master ports which allow parallel access to the different data structures that are needed by the

algorithm such as the hash tables, Bloom filters, and the dynamic FIFO buffer. The basic architectural limitation of this baseline design is that each one of these memory transactions is limited to a width of 128 bits, which is due to the original embedded platform target.

3.2 Preprocessing Phase

The Preprocessing Stage is an important aspect of the kernel's all-FPGA design and it is responsible for converting the raw and unstructured data graph into a set of highly optimized data structures in memory which are optimized for the WCOJ algorithm. The preprocessing phase is implemented as a complete pipelined design with dataflow functions that implements a two-pass algorithm on the input graph data to correctly size and sort the final data structures to optimize efficiency of the subsequent Multiway Join Phase. The output of this preprocessing stage is a set of two-level hash tables and the associated Bloom filters, one for each relation (i.e. each type of edge) which are described by the query graph and stored in the off-chip DRAM of the embedded FPGA[3].

3.2.1 Two-Pass Algorithm

Pre-allocating space for the hash tables according to the worst-case scenario would be inefficient in case of power-law graphs, therefore the preprocessing pipeline executes a two-pass algorithm similar to a counting sort [3] which is detailed as:

1. **Pass 1: Collision Counting:** In the first pass, the kernel traverses every edge of the data graph and it computes the hash values of the source and destination vertices of every edge. These hashes are used to identify which cell in the two-level hash table the edge belongs to, which allows the kernel to simply increment a counter associated with that cell instead of storing the edge. The pass basically constructs a histogram of edge distribution across all hash buckets for all relations, and at the end of this pass the kernel has an exact count of the number of edges that will exist in each bucket.
2. **Pass 2: Offset Calculation and Edge Storage:** Pass 2 starts by computing a prefix sum on the number of collision counts generated in Pass 1, and this calculation converts the raw counts into a table of memory offsets such that each entry now points to the starting address of a pre-allocated and efficiently sized memory region for its corresponding edge bucket. The accelerator then runs a second time and streams through the data graph edges again, and for each edge it re-computes the hash values, retrieves the current write offset of that bucket, stores the edge at that memory offset in off-chip memory, and

then it finally increments the offset. This process scatters all the relevant edges of the graph to their final and sorted positions in memory.

3.2.2 Core Data Structures

The output of the preprocessing phase is a set of data structures which are designed to allow near-constant time access during the next join phase.

- **Two-Level Hash Table:** The primary data structure is a two-level hash table that is characterized by two hash width parameters h_1 and h_2. For a given relation $R(u_1, u_2)$, an edge (v_1, v_2) from the data graph is mapped by using two hash functions, $H_1(v_1)$ and $H_2(v_2)$, where v_1 is the indexing vertex. The first-level hash, H_1 partitions the relation into roughly organized hash buckets, and then the second-level hash H_2 divides each of the hash buckets into sorted and organized cells. The hash table is a matrix-like data structure that is stored in off-ship and it contains the memory offsets of Pass 2 and it allows the kernel to access the range of edges that correspond to a specific hash pair $(H_1(v_i), H_2(v_j))$ using two memory accesses only.
- **Bloom Filters:** The kernel also generates an associated Bloom filter for each of the first-level hash buckets (i.e. for each row in the hash table) in parallel with the second pass. A Bloom filter is a probabilistic data structure that allows for a space-efficient set representation and it supports the testing of approximate set membership with the property that false positives are possible but false negatives are not. In the LESS kernel's architecture, the Bloom filter for a given bucket stores a compact representation of all indexed vertices that are contained within that bucket, and these filters are then used in the Multiway Join Phase to perform a fast, approximate set intersection which allows the kernel to eliminate a significant portion of the search space before performing the more time consuming full edge data access from memory.

3.3 Multiway Join Pipeline

The Multiway Join Phase is the main execution component of the subgraph isomorphism kernel which executes the main subgraph matching algorithm, and it performs an iterative and vertex at-a-time enumeration of all the valid embeddings. This phase is designed as a deeply pipelined and looping dataflow architecture where partial solutions are continuously read from the dynamic FIFO, extended by the pipeline and then written back to the FIFO, and this iterative process is repeated until either all the query vertices have been successfully mapped to create a complete embedding, or the FIFO of partial solutions becomes empty which

means that no further matches can be found. The actual pipeline consists of four main functional stages which include Propose, Intersect, Extract, and Verify.

1. **Propose Stage:** The multiway join pipeline starts with the Propose stage which is responsible for identifying the most constrained set of candidate vertices for the next query vertex that is being matched. For a given partial solution p, this step first determines the next vertex u in the matching order, and then it identifies all the relations R that u is a part of. The main operation in this step is to find the smallest candidate set among all these relations, and this is done by computing the size of each candidate set by either reading the pre-computed size of an indexing set or by comparing two offsets in the hash table to determine the size of an indexed set, and once the smallest set (the minimum set) is identified, the Propose stage loads its component vertices from off-chip memory and then streams them to the next stage. The Propose stage implements a type of find-smallest-first rule which is the basis of the WCOJ algorithm because it ensures that the complexity of the later intersection and verification steps is proportional to the size of the smallest set [2].
2. **Intersect Stage:** The Intersect stage performs a quick and approximate set intersection to eliminate the candidate vertices that are sent by the Propose stage, and for each candidate vertex v from the minimum set, the Intersect stage checks whether or not it is contained in all other relevant relations. This check is not performed by reading complete sets of edge lists but by querying the pre-computed Bloom filters, and a candidate v is discarded if the Bloom filter for any of the other required sets shows that v is not present, and since Bloom filters can produce false positives, this intersection is only approximate such that it ensures that no valid candidates are discarded but it may allow some invalid candidates to pass through. This trade-off is necessary for performance as it replaces a large number of costly random memory accesses with few and very efficient sequential Bloom filter reads [3].
3. **Extract Stage:** The candidates which are remaining after the approximate intersection are then sent to the Extract stage, and the main function of this stage is to perform the conversion from the hash values back into the vertex IDs and to filter out duplicates that might occur due to hash collisions, and then convert the incoming vertices into a proper set.
4. **Verify Stage:** The last main functional stage of the multiway join is the Verify stage which identifies the uncertainty created by hash collisions and the false positives from the Bloom filter intersection, and on every candidate vertex v which has already passed the previous filters, this stage performs the final correctness checks:

- **Homomorphism Check:** The Verify stage first verifies that the candidate vertex v is not already present in the current partial solution p and by doing so it enforces the injectivity constraint of isomorphism.
- **Edge Verification:** It then performs a complete check to confirm that the vertex v is connected to all the necessary vertices in the partial solution, and this is done by accessing the two-level hash tables to read the small, final edge lists and searching to find the exact edges that are required for connectivity.

Candidates that pass both of these checks are then considered valid extensions, and for each valid extension a new and larger partial solution is generated and written back to the dynamic FIFO so that it can either be processed in the next iteration of the multiway join, or it can be written to the output in the case when a complete embedding is identified.

3.4 Core Data Structures and Optimizations

The optimized and complete mapping of the memory-bound WCOJ algorithm to an FPGA architecture is performed based on the combination of a series of well optimized data structures and hardware optimizations which are designed and implemented to utilize properties of data locality, hide the high latency of off-chip memory and manage large size of intermediate results generated during the join process. The main architectural contributors in the baseline design are the AdjHT struct, a custom multi-level cache and a dynamic FIFO for managing partial solutions.

3.4.1 Adjacency Hash Table Descriptor

The two-level hash table and its associated edge lists are saved in off-chip memory, but all metadata that is required to access them is stored on-chip in a small structure called AdjHT (Adjacency Hash Table), where for each relation this structure contains the necessary pointers and counts that are needed by the pipeline to access them. The structure mainly includes:

- $start\_offset$: The base address in memory where the two-level hash table (the array of offsets) of this relation starts.
- $start\_edges$: The base address in memory where the compressed and sorted array of edges of this relation starts.
- n_edges: The total number of edges which have been stored for this relation.

By storing this metadata on-chip the kernel is able to perform the calculation

for the exact address for any hash table lookup or edge list access without any additional memory access latency.

3.4.2 Custom Cache Implementation

The baseline architecture's performance is fundamentally limited by memory latency and bandwidth, and to optimize this a custom two-level parameterized cache is implemented into the architecture [3] in order to mitigate the performance gap between off-chip memory and on-chip logic and optimize the memory access pattern of the WCOJ algorithm.

- **Exploiting Spatial Locality:** The preprocessing phase creates a large amount of spatial locality into the final data structures by sorting edges based on their vertex hash values, and after that when a step of the multiway join phase, Verify for example, might need to verify the existence of a set of multiple edges present inside the same hash bucket, these edges now exist physically closer in memory and the cache takes advantage of it by fetching a complete cache line (composed of a series of multiple 128-bit words in the baseline design) when the first miss occurs. Now, multiple data requests within that same line can be served to the kernel directly from the low-latency on-chip BRAMs which have low latency, helping the kernel to avoid performing off-chip memory accesses which have high latency.
- **Multi-Instance Configuration:** The baseline kernel design instantiates a set of multiple independent cache objects where each cached is customized according to a specific access pattern for the multiway join pipeline. For example, the cache that is used for reading the minimum set can be configured differently from the one that is used for offset table lookups and final edge verification. This parametric cache design allows for small optimizations so that different aspects of the algorithm can be optimized with different parameters of the independent caches such as size, associativity or storage type depending on the cache size (BRAM or URAM).

3.4.3 Dynamic FIFO for Partial Solution Management

The WCOJ algorithm expands in breadth-first style as it processes vertices one at a time, and this property can cause combinatorial explosion in the size of intermediate partial solutions, which can result in a very large set of solutions and storing them is an important architectural challenge. An on-chip FIFO would fill up very quickly and result in stalling the whole pipeline whereas only using off-chip memory would add latency to the main feedback loop of the algorithm.

The baseline accelerator solves this with a dynamic FIFO which is a solution that smartly handles incomplete solutions in on-chip and off-chip memory [3]. The

FIFO is dynamic because its effective capacity is not a fixed, compile-time constant, and it is different from a standard hls::stream that synthesizes to a fixed-depth FIFO using on-chip BRAMs, because the effective size of the FIFO can dynamically grow to accommodate a large number of intermediate solutions, limited by size of the results buffer in off chip memory. Under normal operation when the number of partial solutions is low, the dynamic FIFO is used in the same way as a conventional, low-latency on-chip hls::stream which have single-cycle throughput and partial solutions are written to and read from the FIFO to make sure that the multiway join pipeline runs at full speed.

Whenever the number of items stored in the FIFO exceeds the predefined FIFO depth, the FIFO memory management unit packs partial solutions into wide memory words and burst writes them to a pre-allocated circular buffer in off-chip memory and the pipeline is simultaneously fed from the on-chip data of the FIFO. After the on-chip buffer has been filled up, the kernel will then start re-filling it by burst reading the buffer from off-chip memory, and this helps the kernel to use a high-performance buffer to hide the memory access latency in this process.

The primary limitation of this dynamic FIFO is its fixed, compile-time defined capacity for the off-chip buffer, and although the space is large (hundreds of megabytes) it is still limited and for complex query graphs that result in a very large intermediate result set, it is possible to overflow this space which would lead to the kernel failing. However, for a wide range of practical workloads, this mechanism provides a reliable working solution to the management of memory for the WCOJ algorithm.

Chapter 4

Memory Subsystem Performance Analysis

4.1 Performance Characterization of WCOJ on FPGAs

The baseline accelerator architecture, as described in the previous chapter, successfully implements the Worst-Case Optimal Join (WCOJ) algorithm onto a deeply pipelined FPGA architecture. An important characteristic of the WCOJ paradigm is its algorithmic efficiency, which in theory bounds the size of intermediate results to limit computational complexity of combinatorial complexity and it moves the main computational burden to data movement to and from memory, because all the main operations of the multiway join pipeline like scanning the minimum set, checking Bloom filters and verifying edge connectivity are all series of accesses to the different data structures which exist in off-chip memory.

As a consequence, it is hypothesized that the overall throughput of the accelerator is not limited by the available on-chip computational resources but rather the accelerator is memory-bound and the bottleneck is now shifted to memory, and the rate at which the kernel can process embeddings is now directly dependent on the available bandwidth and latency of the external memory subsystem. In such a memory-bound system, the physical width of the memory interface and the memory technology (e.g., DDR4 or HBM) become the main factors which bottleneck the performance that can be achieved by the accelerator.

This hypothesis serves as the motivation for the work that is presented in this chapter, because before any architectural modifications are made to the kernel, it is important to quantitatively define the achievable performance of the target memory systems and a detailed analysis of the available memory bandwidth and latency

is required to establish an empirical upper limit on the memory's performance, and also to justify the modifications to the datapath of the kernel. The following sections detail the methodology and the results of a detailed benchmark study which is performed to profile the memory subsystems of modern data center FPGAs to provide data and guide the architectural modifications for the accelerator to adapt it to data center FPGAs and improve its performance.

4.2 Benchmark Methodology

To qualitatively test the memory-bound kernel hypothesis and to establish a performance baseline for, two different microbenchmark kernels were developed using Vitis HLS, namely a **bandwidth profiler** and a **latency profiler**, adopting the methodology proposed by Lu et al. [4]. These kernels were designed to isolate and measure the two main properties of the memory subsystem using different access patterns to provide a detailed understanding of performance of the memory subsystems of target data center platforms, and both kernels were designed to run on a 512-bit wide data path (axi_word) to utilize the complete physical interface of the Alveo platforms.

4.2.1 Bandwidth Profiling Kernel

The objective of the bandwidth profiling kernel is to measure the maximum achievable data transfer rate by saturating all the parallel memory channels of the target FPGA.

Kernel Architecture

The kernel was designed as a high-throughput data movement kernel which is capable of parallel execution. To measure the aggregate bandwidth of the target FPGA, multiple Compute Units (CUs) were instantiated, and each CU's AXI master ports were mapped to a unique HBM pseudo-channel or DDR channel using Vitis linker directives. This configuration is demonstrated by [4] to be effective for saturating parallel memory channels as it ensures that the CUs are able to run in parallel without contesting for the same physical memory resources and it allows the benchmark to correctly measure the actual per channel throughput and aggregate throughput of the target FPGA.

The kernel also has outstanding read and write request FIFOs with a depth of 16 elements, and it is divided between two Super Logic Regions (SLRs) to manage routing congestion.

Operational Modes

The kernel supports three major operational modes OP_READ, OP_WRITE, and OP_COPY for evaluating both unidirectional and bidirectional data movement, as well as two major access patterns to model the different types of memory accesses that exist in the WCOJ algorithm.

- **Sequential Access:** The sequential mode allows the kernel to access a large, contiguous block of memory sequentially and perform memory word wide burst transfers. This pattern is expected to measure the ideal, peak sequential throughput of the memory system, which is representative of workloads that have high spatial locality, which also exists in the LESS kernel WCOJ algorithm such as scanning Bloom filters or large candidate lists.
- **Random Strided Access:** This mode allows the kernel to perform a series of non-contiguous memory accesses and step through memory at a configurable stride. This pattern emulates the less predictable access patterns of WCOJ algorithm that occurs during the Verify stage when the accelerator might require accessing different hash table entries.

Measurement

The performance of the kernel is measured by the host by timing the total execution duration of the kernel for a large number of iterations and the overall throughput is then obtained by dividing the sum total of the bytes transferred ($Data\ Size \times Iterations \times Number\ of\ CUs$) by the elapsed time.

4.2.2 Latency Profiling Kernel

The objective of the latency profiling kernel is to accurately measure the round trip time for a single memory access from the perspective of the kernel logic, by isolating it from external software overhead.

Kernel Architecture

This kernel architecture is made up of a dataflow architecture, similar to the work done in other microbenchmark kernel studies [4], and the kernel has three different concurrently executing modules to separate the measurement logic from the memory access logic, and the kernel is designed with read and write outstanding FIFOs with a depth of just 1 to disable bursting of memory accesses so that the latency of a single memory access can be isolated. The three modules communicate using hls::stream FIFOs, and they are:

- **Accessor Module:** This module holds the main memory access loop, and it is responsible for sending a start signal on a stream just before it starts its first memory operation and then it sends a stop signal after its final operation is completed.
- **Timer Module:** This module contains a basic, free-running cycle counter which begins counting when it receives the start signal from the accessor module and it stops counting when it receives the stop signal.
- **Collector Module:** The collector module is the final module that reads the final cycle count from the timer and the total number of accessess performed by the accessor module, and then it writes these values to AXI-Lite registers for the host.

This dataflow-based design separation and use of different modules helps make sure that the latency measurement only includes the execution time of the memory operations and removes software overheads.

Calibration Mode

An important feature in this kernel is a calibration mode, which when enabled, causes the Accessor module to execute the same loop structure and signaling protocol but it doesn't include the physical memory access instructions. The resulting cycle count in this mode gives us the overhead of loop control and inter-module stream communication, and to obtain the actual memory access latency this overhead cycle count is then subtracted from the total measured cycles during a memory test run, and then the result is divided by the number of accesses. of the total cycles measured are divided by this number.

Measurement

The latency profiling kernel also includes the same operational modes as the bandwidth profiling kernel and this benchmarking design provides a clock-accurate measurement for the average latency of a single memory access, and it separates the memory benchmark from non-deterministic host-side timing and operating system scheduling overheads because by using this technique, the benchmark can easily determine latency based on the result from the Timer module and the kernel's operating clock frequency by directly reading the cycle count and number of memory accesses from the AXI-Lite registers in the kernel.

4.3 Experimental Setup

In order to perform a detailed comparison between the HBM and DDR4 memory technologies, the benchmark kernels were built and tested on two different AMD

Alveo data center accelerator cards, the development was done on Vitis HLS and the host applications were built using the Xilinx Runtime (XRT) library.

4.3.1 Hardware Platforms

HBM Platform (AMD Alveo U55C)

The HBM performance was profiled using an Alveo U55C accelerator card, which has a single FPGA die and 16GB of HBM2 memory, which is connected with the FPGA as 32 independent 512 MB pseudo-channels, each of which have a 512-bit wide AXI interface per memory bank. The bandwidth benchmark was configured to instantiate 16 Compute Units (CUs) and the two memory ports of one CU were connected to two different banks of the 32 HBM banks available so that the memory subsystem could be fully saturated.

DDR4 Platform (AMD Alveo U250)

The DDR4 performance was benchmarked on an Alveo U250 accelerator card which has 64 GB of DDR4 memory spread across four independent memory channels (banks), and the bandwidth benchmark was configured similar to the HBM setup by using two CUs and the memory ports of each CU were assigned to a different pair of the four available DDR banks.

The kernel frequency of both platforms was configured to a target frequency of 300 MHz for both the bandwidth and latency tests, which is the highest available clock frequency for HLS kernels on the tested Alveo platforms to utilize the memory subsystems at maximum capacity.

4.3.2 Benchmark Parameters

The profiler kernels were executed with different parameters to characterize performance, and the main parameters are:

Data Sizes

In bandwidth testing, the size of the total data transfer for sequential and random stride access patterns (64B stride) was set at 256 MB across all 16 control units and the test was run for 1000 iterations, both of which were large enough to ensure that the measured performance results represent steady-state throughput, minimizing the impact of kernel startup and shutdown and other control overheads.

The total processed data for sequential and 64-byte random strided tests was 250 GB, while the data size for random stride access tests for a larger stride of

2 MB was 512 MB with 10,000,000 iterations with a total processed data size of 152.58 GB for each test.

The latency benchmark data size was set at 256 MB for both sequential and random stride access tests.

Access Patterns

The bandwidth and latency benchmarks evaluated both sequential and random-strided access patterns with different strides (64 Bytes, which is a single cache line) and very large strides (2 MB) to test memory performance with different levels of memory access locality.

AXI Interface Configuration

The m_axi interface pragmas in the HLS kernels were configured with different depths for outstanding read and write request FIFOs for the bandwidth and latency profiling kernels according to the nature of the benchmark, because the bandwidth benchmarks require high depth FIFOs for maximum performance while the latency benchmarks require FIFOs with depth 1 to evaluate true memory latency so that the FIFOs don't hide memory latency.

The XRT host application for both kernels allocate the device buffers for the connected memory banks, manage the execution of the kernels and timing the kernel operations to calculate the final throughput and latency metrics.

4.4 Benchmark Results and Analysis

The bandwidth and latency profiling kernels were executed on the AMD Alveo U55C (HBM) and U250 (DDR4) platforms to quantitatively evaluate the performance of the memory subsystems. In this section, the results have been summarized and they provide a clear and experimental basis for the architectural adaptation of the subgraph isomorphism accelerator. The analysis focuses on aggregate throughput and average latency of a single memory operation under both sequential and random memory access patterns.

4.4.1 Aggregate Bandwidth Analysis

The aggregate bandwidth tests were configured to utilize all the available memory on each platform (32 HBM pseudo-channels on the U55C and 4 DDR4 channels on the U250) to measure the theoretical maximum data transfer rate of the platforms.

Sequential Throughput

The WCOJ algorithm has a number of important operations such as scanning the minimum candidate set during the Propose stage and reading Bloom filters during the Intersect stage, all of which have sequential memory access pattern.

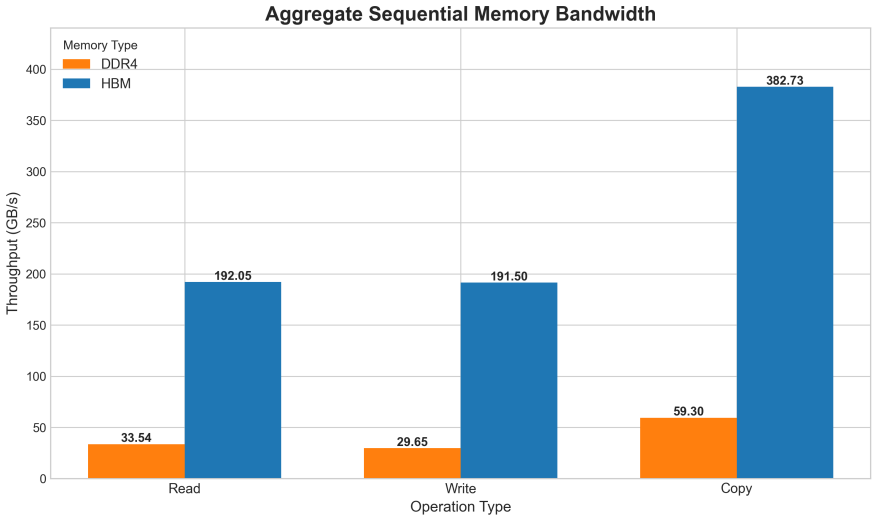

Figure 4.1: Aggregate Sequential Bandwidth (HBM vs. DDR4)

The HBM platform has a major advantage as seen in Figure 4.1, and for sequential copy operations, the Alveo U55C achieved an aggregate throughput of approximately **382 GB/s** while the Alveo U250 platform under the same conditions provided a maximum throughput of ~**59 GB/s**, and this represents a ~**6.5x improvement** in peak memory bandwidth for the HBM-based platform over the DDR-based platform.

The individual throughput values for read, write and copy operations are detailed in Table 4.1 which were tested with the same bandwidth profiling kernel and the same parameters, which confirms that the advantage of higher bandwidth is dependent on the type of memory subsystem and is consistent across different memory access patterns.

Table 4.1: Sequential Aggregate Bandwidth Benchmark Results.

Operation Type	HBM BW (GB/s)	DDR4 BW (GB/s)	Performance Ratio (HBM/DDR4)
Read	192.05	33.54	5.73x
Write	191.50	29.65	6.46x
Copy	382.73	59.30	6.45x

A more detailed analysis also involves looking at the **per-channel bandwidth** of both memories to get more information about the capabilities of the memory subsystem. This metric provides the efficiency of a single memory interface and allows for a more direct comparison of throughput for each independent channel without considering the overall number of channels, and the per-channel throughput is calculated by dividing the measured aggregate throughput with the number of active memory channels used by the benchmark (32 channels in HBM on U55C and 4 channels in DDR4 on U250).

Table 4.2: Per-Channel Sequential Aggregate Bandwidth Analysis.

Operation Type	Memory Type	Aggregate Throughput (GB/s)	Number of Channels	Per-Channel Throughput (GB/s)
Read	HBM	192.05	16	12.00
	DDR4	33.54	2	16.77
Write	HBM	191.50	16	11.97
	DDR4	29.65	2	14.82
Copy	HBM	382.73	32	11.96
	DDR4	59.30	4	14.83

Table 4.2 shows the result of this analysis, which shows the interesting finding that for sequential operations a single DDR4 channel provides more throughput than a single HBM2 pseudo-channel, for example, in the copy operation a single DDR4 channel had a sustained throughput of approximately **14.83 GB/s** while the HBM pseudo-channel achieved a throughput of **11.96 GB/s**. These results are also consistent with the characterization work performed in other studies [4].

The architectural differences in the memory interfaces and their connection with the FPGA explain these results. The four DDR4 memory banks of the Alveo U250 and the 32 HBM pseudo-channels of the Alveo U55C all connect to the FPGA through a wide 512-bit AXI interface, allowing both interfaces to perform burst transfers during sequential memory access operations, but the higher clock frequency of the DDR4 interface allows it to transfer more data per cycle on a single channel than the interface for a single HBM pseudo-channel, resulting in slightly worse per-channel throughput for the HBM memory interface.

However, this result does not reduce the overall superiority of HBM for high-performance computing, but rather highlights the fact that the architectural advantage of HBM isn't in the speed or throughput of a single channel, but in the massive parallelism that exists in the HBM interface, and by providing eight times the number of independent memory channels, the HBM subsystem can service a much greater number of parallel memory requests and provide significantly

better aggregate throughput. This performance result also validates the multi-CU benchmark methodology because it shows that the aggregate performance advantage of the HBM interface is proportional to the number of channels and it also supports the main hypothesis that a memory-bound accelerator needs to be designed with a large number of parallel memory ports and a wide internal datapath to effectively take advantage of the massive parallelism that the HBM memory interface provides.

Random-Stride Throughput

The random-stride access pattern is arguably more important for the WCOJ algorithm, since it mirrors the memory access behavior of the Verify stage where the kernel needs to perform lookups into the hash tables to determine whether an edge is connected or not, and the performance in these conditions shows the capability of the memory subsystem to handle computations which have poor spatial locality in memory. To measure this, the bandwidth profiling kernel tests the memory's performance with a small 64-byte stride which represents frequent jumps between nearby cache-line sized data, as well as with a relatively larger 2 MB stride, which is a worse case scenario of accessing farther memory regions.

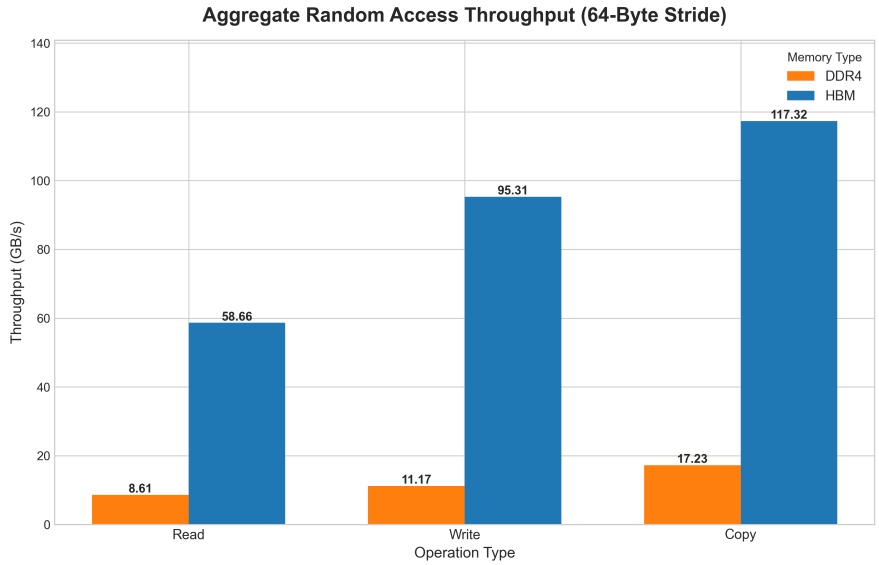

Figure 4.2: Aggregate Random Access 64B Stride Bandwidth (HBM vs. DDR4)

Figure 4.2 visualizes the performance of random read, write and copy operations using a 64 bytes stride. In terms of random copy operation, the HBM platform had a overall throughput of approximately **117 GB/s** compared to the DDR4 platform that had a maximum of approximately **17 GB/s**, which represents a

~6.9x performance gain in case of HBM. This result shows that even when memory access is not sequential or possible to be executed in memory bursts, the parallel architecture of HBM interface still provides a much higher throughput than DDR4 in the case of non-contiguous access patterns because of the parallel architecture.

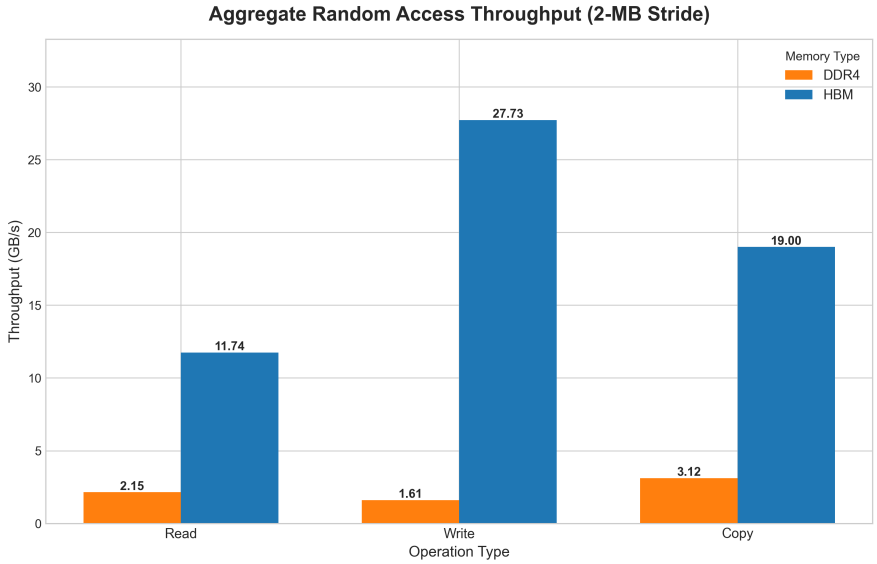

Figure 4.3: Aggregate Random Access 2-MB Stride Bandwidth (HBM vs. DDR4)

The random-stride access benchmark was repeated with a relatively larger 2 MB stride to further reduce memory spatial locality, minimize data reuse and avoid prefetching and caching optimizations, and HBM still demonstrated significantly higher throughput where in the random copy operation, the HBM platform maintained an approximate throughput of **19 GB/s** whereas the throughput of the DDR4 platform reduced to about **3 GB/s**, which is a performance ratio of about **6.1x**. The results of this second test are visualized in Figure 4.3, and the detailed benchmark results are given in Table 4.3.

The main insight from this analysis is that although both memory technologies suffer from performance with a reduction in spatial locality (as the stride size increases), HBM's throughput still significantly outperforms DDR4 due to its parallelism from 32 independent pseudo-channels, which makes it more well-suited for handling a large number of random memory requests. This high performance benefit on a wide range of access patterns validates HBM as the better memory technology for implementing memory-bound graph algorithms, and it also justifies the need to have an accelerator architecture that is designed to be able to take full advantage of the parallelism provided by HBM.

Table 4.3: Comparison of Aggregate Random-Stride Memory Bandwidth.

Operation Type	Stride Size	HBM BW (GB/s)	DDR4 BW (GB/s)	Performance Ratio (HBM/DDR4)
Read	64 Bytes	58.66	8.61	6.81x
	2 MB	11.74	2.15	5.46x
Write	64 Bytes	95.31	11.17	8.53x
	2 MB	27.73	1.61	16.91x
Copy	64 Bytes	117.32	17.23	6.81x
	2 MB	19.00	3.12	6.09x

4.4.2 Latency Analysis

The average access latency is another important measure that characterizes the responsiveness of the memory subsytem to individual, non-contiguous access requests. In the case of memory-bound algorithms with random memory access patterns such as the WCOJ algorithm, low latency is necessary to avoid pipeline stalls and to achieve high throughput. The latency profiling kernel was used to perform a clock-accurate measurement of the round-trip latency of one 512-bit access with the help of the dataflow-based timer and accessor modules.

The results for the benchmark are presented across Figures 4.4, 4.5, 4.6, and all the results are also summarized in Table 4.4, and they show the detailed performance differences between HBM and DDR4 interfaces in terms of latency.

Sequential Access Latency

The sequential access pattern results shown in 4.4 measure HBM sequential read latency of **195.02 ns**, which is almost identical to its random-access latency, and is also similar to DDR4 latency of **206.80 ns**. This is not the total latency of a continuous burst, which would be so very low, but instead it is the measurement of average latency for the round-trip time of single, non-overlapping memory requests over multiple iterations of memory requests, where each memory operation is initiated independently even in this case where the targeted memory addresses are contiguous. This is done by having outstanding read and write FIFOs of size 1, and therefore the benchmark measures the latency of a single access. The write performance is one of the more important findings of this benchmark, as a sequential write latency was found to be **96.66 ns** in the HBM subsystem, which is much lower than the **162.48 ns** of DDR4, showing that **HBM write speeds are 1.68x faster**, even with single-access.

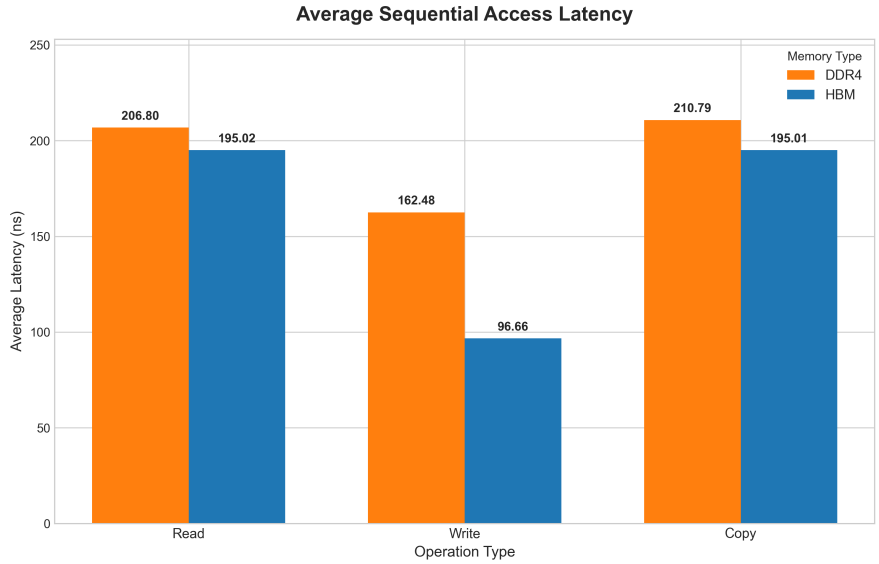

Figure 4.4: Comparison of Average Sequential Access Latency (HBM vs. DDR4)

Random-Stride (64B) Access Latency Test

The 64-byte stride test, shown in Figure 4.5 is the most representative test for the comparing performance of both technologies for frequent, non-contiguous, memory lookups in the WCOJ algorithm.

In terms of read performance, the HBM platform provided a read latency of **195.02 ns** while the DDR4 platform provided **206.80 ns**, this confirms that for isolated random reads, both memories are competitive in performance and HBM shows a negligible advantage, supporting the conclusion that the main architectural advantage of HBM is its massive parallelism to achieve high bandwidth rather than a high single request read speed.

The write latency advantage of HBM is is again highlighted in this test, where HBM achieved a write latency of **109.99 ns** which compared to **177.26 ns** for DDR4 provides a **1.61x performance improvement**. This can be a key advantage for various parts of the subgraph isomorphism kernel such as the dynamic FIFO, which has to write contents to off-chip memory in a series of write operations when there is a large number of intermediate partial solutions, and having a lower write latency provides HBM the advantage to lower the performance cost of such critical functions.

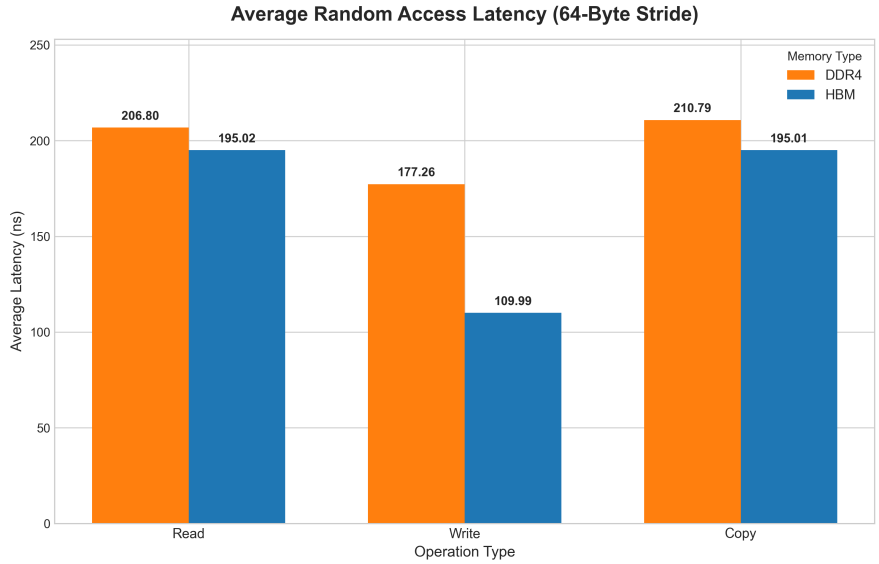

Figure 4.5: Comparison of Average Random Access Latency at 64B Stride (HBM vs. DDR4)

Random-Stride (2MB) Access Latency Test

The results for the average random latency test with 2 MB stride are seen in Figure 4.6, and the test models a random access pattern with minimum spatial locatily of data to force the memory controller to access farther memory regions, and it tests the performance of the memory in cases which cause DRAM page misses.

The read operations in both HBM and DDR4 platforms suffered an increase in latency compared to the 64B stride test, with average latency values increasing to **232.32 ns** in HBM and **234.82 ns** in DDR4. This increase can be explained by the fact that there is a high probablity of a row-buffer miss (or page miss) in the DRAM, in which case the memory controller will need to close the currently active memory row and activate a new one, resulting in higher latency.

However even in this scenario of a higher stride for random access, the write latency of HBM was still significantly lower at **111.47 ns** than that of DDR4 at **179.57 ns**, resulting in a **1.61x performance increase** for HBM writes, showing that the HBM interface has superior write performance even in the case of random access patterns with poor data spatial locality and farther memory regions.

The detailed results of the latency benchmark are provided in Table 4.4, and these findings empirically prove the hypothesis of the thesis. The memory performance characterization tests show that while the single-access read latency of HBM is the same as DDR4, its write latency is much superior in both sequential and random

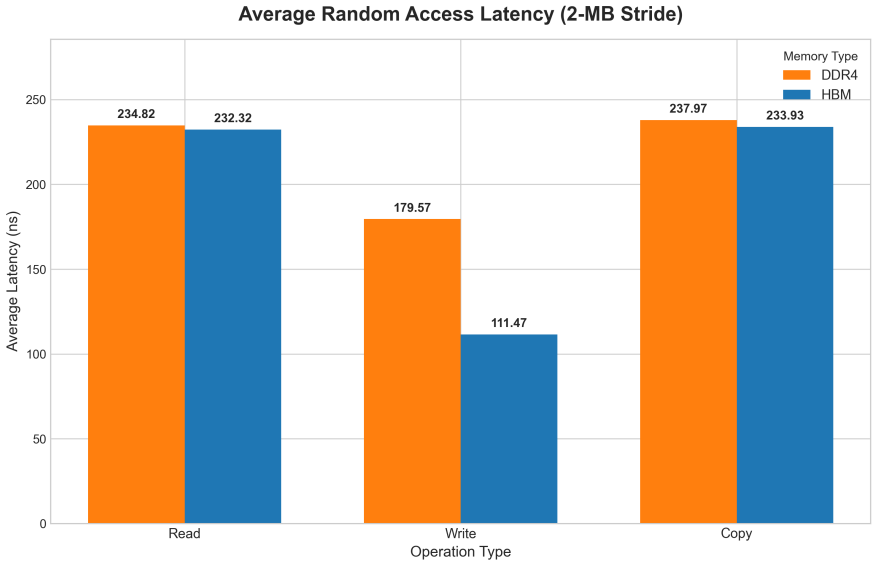

Figure 4.6: Comparison of Random Access Latency at 2MB Stride (HBM vs. DDR4)

memory access patterns, which is a significant benefit for memory-bound algorithms. Most importantly, once the latency benchmark results are considered along with the bandwidth analysis, it becomes clear that to improve the performance of the LESS kernel built for an embedded FPGA platform and adapt it for a data center FPGA, its datapath needs to be adapted for the wider physical bus and the kernel needs to be designed to maximize the number of parallel memory requests and fully utilize the massive parallelism that is offered by the HBM interface.

Table 4.4: Comparison of Average Access Latency for 512-bit Operations.

Operation Type	Access Pattern	HBM Latency (ns)	DDR4 Latency (ns)
Read	Sequential	195.02	206.80
	Random (64B Stride)	195.02	206.80
	Random (2MB Stride)	232.32	234.82
Write	Sequential	96.66	162.48
	Random (64B Stride)	109.99	177.26
	Random (2MB Stride)	111.47	179.57
Copy	Sequential	195.01	210.79
	Random (64B Stride)	195.01	210.79
	Random (2MB Stride)	233.93	237.97

Chapter 5

High-Bandwidth Kernel Implementation and Performance Evaluation

The performance analysis in the previous chapter empirically established that HBM data center FPGAs provide much higher throughput capability than conventional DDR4 memory subsystems, and this capability can potentially provide improvement for the primary memory bandwidth bottleneck of the WCOJ-based subgraph isomorphism algorithm. This chapter describes the architectural and implementation work to utilize this potential by migrating the baseline 128-bit accelerator to a 512-bit, HBM-based implementation to be used on the AMD Alveo platform. The main methodology for this adaptation involves an experimental HLS-based approach for datapath widening, and a detailed evaluation to validate the performance improvements from this adaptation.

5.1 Migration from Embedded to Data Center Design Flow

The initial and most important step in this work was the process of adapting the baseline accelerator from the embedded design flow to the Vitis data center development flow using Xilinx Vitis HLS 2024.1, which involves redesigning the host-kernel interaction model and a different management of hardware resources.

5.1.1 Vitis Flow and XRT Host Integration

The original embedded design depended on a different software environment and migrating to a PCIe-based data center accelerator card required the development of a new host application, which is done using the Xilinx Runtime (XRT) library. The XRT API is now used by the host application to explicitly control the target device, load the FPGA binary (.xclbin), and get a handle to the subgraph isomorphism kernel. The XRT API also provides buffer management functionality to perform all memory allocations on the FPGA's off-chip HBM through XRT Buffer Objects (xrt::bo), and the host is responsible for creating these buffers for all the kernel's memory spaces, including the main hash table buffer (htb_buf), the bloom filter buffer, and the large dynamic FIFO space (res_buf), providing a standardized way for allocating memory in specific HBM banks and for managing data transfers between the host and the device.

Another difference that needs to be considered is that the embedded platform has a shared-memory design, whereas the data center flow requires data synchronization, and the XRT host application is created with this consideration which uses sync() API calls for transfering input graph data from the host memory to the device's HBM before kernel execution starts.The host then configures and launches the kernel using an xrt::run object, which is a high level XRT API for managing kernels and it abstracts away handing of lower level control registers. The host passes all scalar parameters such as graph dimensions and hash configuration values (h1, h2) to the kernel via the AXI-Lite control interface, and then upon completion of the kernel the host reads back the final results and debug counters from the kernel's AXI-Lite registers.

An important aspect of this integration is the Vitis linker configuration. The top level kernel is designed with multiple m_axi interfaces to provide parallel memory access and maximize bandwidth. The Vitis linker configuration file (.cfg) is used to explicitly map each of these logical interfaces into a separate physical HBM bank on the FPGA, and by assigning different bundles (e.g., cache, fifo, etc.) to different m_axi interfaces, the Vitis linker instantiates separate, parallel memory controllers, preventing conflict between different memory access needs and patterns for different parts of the multiway join pipeline like cache lookups and dynamic FIFO requests, and this is a key technique used for achieving high throughput multi-banked memory systems like HBM.

5.1.2 Data Path Widening using HLS Compiler Directives

The main challenge in adapting the kernel for the high-bandwidth platform was to upgrade the datapath to support a 512 bits physical bus instead of 128 bits. Although a complete redesign of the internal logic of the kernel to natively process 512-bit vectors was initially considered, a more feasible and immediate approach was

adopted for this thesis, and the primary methodology used was to take advantage of powerful optimization capabilities of the Vitis HLS compiler through effective use of the max_widen_bitwidth HLS pragma. This approach was considered because it allows for a quick assessment of the impact of a wider memory bus without significant redesign of the entire kernel's internal logic and it allows to test the hypothesis that the baseline kernel's algorithm is actually memory-bound.

Specifically, *the max_widen_bitwidth=512* directive was applied to all the m_axi interface pragmas in the top-level kernel function, which tells the HLS compiler to perform **automatic bus widening**. The compiler uses this directive to automatically instantiate 512-bit wide AXI interfaces and synthesizes additional hardware logic for data width converters that are placed between the kernel's internal logic and the wide AXI ports. The additional automatically generated logic of the converter performs both data buffering and coalescing, where in case the internal kernel logic issues a sequence of smaller, contiguous memory requests (e.g., four consecutive requests for 128-bit Bloom filters), the converter automatically buffers these requests and once enough data is accumulated for a 512-bit word, it issues a single, wide transaction on the physical AXI bus. This whole process is transparent to the core algorithm logic but it is critical for performance because it ensures to efficiently utilize the physical memory bus and perform a lower number of memory access requests.

This is also a key advantage of this methodology, because it keeps the internal logic of the kernel unchanged and the kernel still operates using its native data types (e.g., 64-bit edges, 128-bit Bloom filters, 32-bit offsets), and it preserves the accuracy and structure of the original and validated algorithm, while the HLS tool manages complexity of the transformation of these narrow logical accesses to the wide physical bus.

This approach produces a hardware architecture where the kernel's core algorithm performs logical memory accesses as it would using a 128-bit memory system, but the synthesized AXI interface intelligently buffers and widens these logical memory transactions to 512 bits. This practical implementation of HLS compiler directives allowed for a rapid and effective migration from an embedded design flow and to directly test the performance impact of utilizing the higher throughput of the HBM subsystem.

The migration from an embedded flow to the Vitis data center flow and creation of an XRT host provides the necessary software and creates a framework for executing and evaluating the accelerator on a high-performance, HBM data center platform.

5.2 Implementation and Performance Evaluation

The effectiveness and accuracy of the architectural migration was validated and the performance impact of the 512-bit datapath was quantified by implementing the the adapted accelerator on a data center FPGA and it was tested using a series of subgraph isomorphism queries. This section details the experimental setup, the methodology used for functional verification, and the final performance results.

5.2.1 Experimental Setup

The accelerator was implemented and tested on an **AMD Alveo U55C** data center accelerator card. This platform was selected because of its **16GB of HBM2 memory**, which is available with 32 independent pseudo-channels, providing the high-bandwidth environment that is targeted by the work in this thesis. The entire project is developed using the **Vitis Unified Software Platform (2024.1)**, with the accelerator logic written in C++ using High-Level Synthesis (HLS) and the host application is developed using the Xilinx Runtime (XRT) library. The kernel is synthesized with a target clock frequency of **300 MHz** to maintain consistency with the previous memory benchmark analysis.

The performance of the accelerator is evaluated using five different real-world graph datasets, **enron**, **github**, **gowalla**, **dblp**, and **wikitalk**, and 30 different subgraph isomorphism queries, all of which are the same that were used to validate and benchmark the baseline design, ensuring a direct comparison between the 128-bit and 512-bit architectures.

5.2.2 Functional Verification

The modified kernel was functionally verified before performing the performance analysis to ensure that the adaptation of the kernel did not introduce any errors into the accelerator's core algorithm. Functional verification was performed by using the subgraph isomorphism enumeration results of the original 128-bit baseline accelerator as a golden reference, and test queries were executed using the enron dataset on both implementation.

For every query in the test set, the final match count produced by the 512-bit HBM accelerator was compared with the counts from the 128-bit baseline kernel golden reference, and in all test cases the 512-bit kernel produced an identical match count, confirming that the adaptation to the Vitis flow and the automated datapath widening performed by the HLS compiler did not affect the functional correctness of the algorithm, and the performance analysis was performed after this successful functional verification test.

5.2.3 Performance Evaluation and Discussion

The performance of the 512-bit HBM accelerator was compared against the 128-bit DDR4 baseline accelerator using the complete set of 30 queries on all the 5 datasets, and the results summarized in Figure 5.1 show that there is a measurable performance improvement that varies depending on the properties of the dataset.

An important initial observation of the experiment was that all query-dataset combinations couldn't be executed successfully on either the baseline or the re-designed accelerator, specifically query 14 on gowalla, queries 28, 29 on dblp and a significant portion of queries on the largest dataset wikitalk (queries 9-29). The main cause of these failures isn't a logic error, but rather it is an architectural limitation of the dynamic FIFO design.

As described in Chapter 3, the dynamic FIFO is designed to manage the large number of intermediate partial solutions which are generated by the WCOJ algorithm which are read from and written to a large pre-allocated buffer in off-chip memory (RESULTS_SPACE). However, the buffer is still limited in size and when the above queries are executed on their large and densely connected graphs, they produce a combinatorial explosion of intermediate results that exceed the capacity of the buffer, leading to FIFO overflow and an incomplete enumeration. Since this limitation is tied to the fixed-size buffer and the constraint is present in both kernels, these specific queries are excluded from the performance comparison, and the subsequent analysis focuses on the set of successfully executed queries on both platforms to provide a direct comparison of the architectural impact of the wider physical bus and high bandwidth HBM memory.

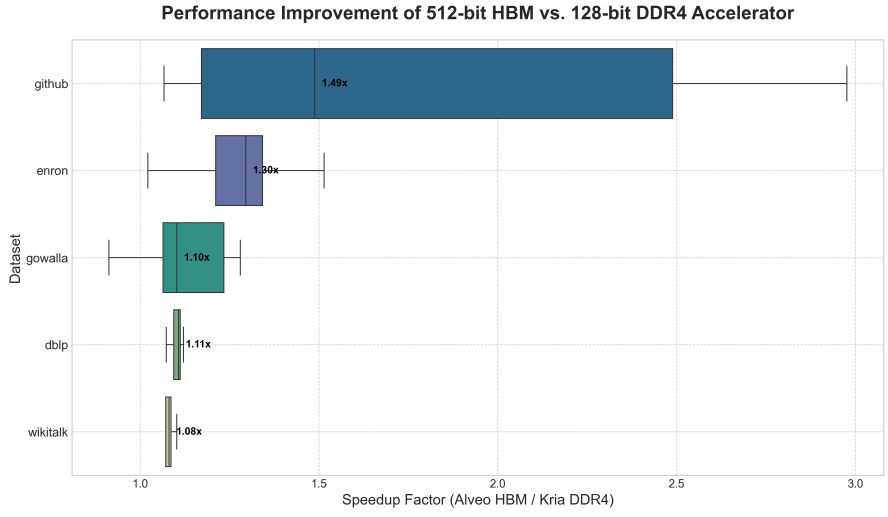

Figure 5.1: Performance improvement distribution of 512-bit HBM accelerator over 128-bit DDR4 baseline for each dataset.

Figure 5.1 shows the distribution of performance improvement (speedup) for each dataset, where on the smaller and sparser enron and github datasets the HBM accelerator with a 512-bit data rate is found to achieve a significant median performance improvement of ~**21.9%** and ~**36.3%**, respectively. This finding proves that the widening of the datapath and the use of HBM provide large benefits when working with workloads that are constrained due to memory bandwidth.

However, as the graph size and complexity increase, the performance improvement is seen to decline, and with the increasingly bigger datasets like gowalla and dblp datasets, the median improvement drops to ~**11.2%** and ~**8.2%**, respectively and on the largest and most complex dataset, wikitalk, the performance is very similar to the baseline accelerator with a median improvement of just ~**5.7%**. This trend of diminishing performance improvement as the dataset complexity increases shows that while sequential bandwidth is an important factor, it is not the only bottleneck and the experimental results show that the speedup just from widening the physical memory bus doesn't directly correlate with the ~6.5x theoretical bandwidth advantage, which can be explained by a number of factors.

The *max_widen_bitwidth* pragma depends on the ability of the compiler to detect contiguous memory access patterns in order to create optimal 512-bit bursts, but this automatic inferring of bursts can be prevented by any complex or data-dependent addressing logic in the kernel and that leads to generation of less-efficient, narrower bus transactions and a reduction in effective bandwidth utilization. This is the most likely cause of the modified kernel underperforming compared to the benchmark results, because although the pragma allows the kernel to utilize the wider physical bus, the kernel is still not utilizing the full parallelism that HBM provides, which would require a major redesign of the kernel to achieve even higher throughput and completely saturate the HBM interface subsystem.

Another possible reason for not achieving the high improvement seen in the benchmarks could be due to random access latency impact, since the benchmark shows that the single-access random read latency of HBM is similar to that of DDR4. The Verify stage of the WCOJ algorithm consists of many random-access lookups into the hash tables, and for complex queries on large graphs like dblp and wikitalk, the execution time might be dominated by these latency-bound accesses rather than sequential bandwidth, which causes the kernel to spend more time waiting for individual data to return from memory, negating most of the benefits of a wider physical bus and making the performance bottlenecked due to memory latency.

Figure 5.2 provides a more detailed comparison of execution time for each query for the 512-bit HBM accelerator and the 128-bit DDR4 baseline. The visualization shows that in most queries and datasets, especially enron and github, the Alveo platform kernel has a consistent, query-level performance improvement, which is a result of the adaptation and redesign, while on complex datasets such as

Figure 5.2: Comparison of execution time for each query for all datasets between 512-bit HBM accelerator and 128-bit DDR4 baseline accelerator.

dblp, the query markers for both accelerators are clustered closely together, and this convergence represents the diminishing returns discussed earlier where the performance bottleneck shifts from a narrow physical bus to a lack of parallelism inside the kernel and random access latency.

Figure 5.3: Performance improvement (Baseline Time/Alveo Time) for each query for all datasets.

Figure 5.3 visualizes the absolute execution times of Figure 5.2 into a direct representation of the performance improvement for each individual query, and it

helps to understand the distributions of performance gains which are summarized by Figure 5.1. The plot shows that although the median speedup can be small on some datasets, a significant number of individual queries still achieve a substantial performance improvement as seen in the case of the github dataset, indicating that the benefit of having a wider datapath are highly dependent on the specific structure of a datagraph and querygraph. The visualization also helps to recognize that the queries with a high speedup factor are most likely to be bandwidth-bound while the ones with a speedup factor close to 1 are likely latency-bound, where the execution time is dominated by random accesses and the wider bus offers minimal advantage. However, in both cases, the kernel can still provide a significant improvement in performance if it is highly parallelized.

In conclusion, the experimental results validate the main hypothesis of this thesis. Adapting the baseline accelerator to a high-bandwidth data center platform and utilizing a wider physical bus provides a significant and measurable performance improvement, particularly on datasets where sequential memory bandwidth is the dominant bottleneck. However, the analysis also provides a more detailed insight that just a wider physical bus is not enough even for high-bandwidth platforms and for more complex graph queries where the random access latency bottleneck is more dominant, a complete redesign of the kernel is required to parallelize the datapath to improve performance. This finding motivates the direction for future work, which is outlined in detail in the final chapter.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis looked at the feasibility of adapting and optimizing memory-bound algorithms in high-performance graph processing, specifically in the context of subgraph isomorphism acceleration on data center FPGAs. The research was informed by the hypothesis that the performance of memory-bound algorithms, such as the ones based on the Worst-Case Optimal Join (WCOJ) algorithm, could be significantly improved, and this hypothesis was validated by migrating a WCOJ subgraph isomorphism accelerator from an embedded FPGA platform with DDR4 memory to a modern data center accelerator platform with HBM. The primary goal of the thesis was to adapt a validated 128-bit WCOJ accelerator to fully utilize the 512-bit wide HBM memory subsytem on the AMD Alveo platform.

To support the motivation behind this redesign, a detailed benchmark study was first carried out to quantitatively characterize the performance of HBM and DDR4, which experimentally established that HBM provided ~6.5x higher aggregate sequential bandwidth and a ~1.6x lower random write latency.

Based on the results from the benchmark study, the baseline accelerator was then successfully migrated to the Vitis design flow and its memory interface was widened to 512 bits by using HLS compiler directives, a technique that allowed for a quick and effective architectural adaptation. The final 512-bit HBM implementation was functionally verified and evaluated against the 128-bit baseline accelerator, and the results demonstrated measurable performance improvement with a median performance gain of upto 36.3%. The analysis also showed that on more complex and larger graphs, the performance gains were relatively lower due to a lack of parallel architecture and other possible bottlenecks, and the experimentation highlighted that while a wide physical bus is necessary, it's not sufficient for utilizing maximum performance from the platform.

In conclusion, this thesis successfully demonstrated that adapting a memory-bound graph accelerator from an embedded platform to a data center platform provides significant and measurable performance gains and the research includes work on migration to the data center design flow, benchmarking of memory subsystems and final implementation of the upgraded accelerator design. It validates the importance of redesigning the accelerator's datapath to match the physical capabilities of the memory system and provides a clear motivation for future work focused on utilizing the parallelism that is available on data center HBM platforms.

6.2 Future Work

The research and implementation detailed in this thesis validate the performance benefits of migrating a memory-bound WCOJ accelerator to a high-bandwidth, HBM-equipped data center FPGA, and the performance analysis also revealed architectural bottlenecks and limitations, providing a clear roadmap for future research and optimization.

6.2.1 Datapath Redesign with a Packed-Instruction Architecture

The max_widen_bitwidth pragma experimentation proved to be an effective first step, however, the experimental results show that this automated approach doesn't completely utilize the HBM interface, especially in latency-bound scenarios. The most significant future work would be the complete implementation of a packed-instruction architecture that was initially evaluated in this research. This architecture is based on the principle of maintaining a 128-bit logical instruction for graph data structures while managing the packing and unpacking of four such instructions into a 512-bit physical memory word, providing the main benefit of reducing memory access requests, because unlike the HLS compiler which relies on inferring and burst-transfering contiguous access requests, this architecture would ensure that every read and write to the HBM is a full 512-bit access request by design.

More importantly, this architecture would enable the on-chip pipeline to be redesigned to operate on wider, 512-bit data vectors, for example, the Verify stage can then be redesigned to process four candidate vertices in parallel, and mwj_findmin stage can process four Bloom filters simultaneously. This architecture would increase the computational density of the kernel and reduce the overall number of required memory transactions, possibly helping to reduce the random access latency bottleneck, providing further performance improvements.

6.2.2 Kernel Replication and Data Partitioning for Parallelism

The current implementation utilizes a single, highly-pipelined accelerator kernel (a single Compute Unit), and the logical next step to this work is to take advantage of the large number of available logic resources of data center FPGAs by instantiating multiple parallel CUs. Two major approaches could be explored, Task-Level Parallelism and Data-Level Parallelism.

A task-level parallelism model uses multiple CUs to process independent queries concurrently, with each CU assigned to a different query. This would significantly increase the overall query throughput of the system. In data-level parallelism, for a single, complex query the graph data could be partitioned across multiple HBM banks and multiple CUs could be instantiated with each CU responsible for processing a specific partition. This would require developing an inter-kernel communication mechanism to merge partial results from different CUs and keeping track of partitions through a separate piece of logic. This implementation is relatively complex but it provides the most potential to significantly reduce the execution time of a single, large and complex query.

6.2.3 Application to Other Memory-Bound Algorithms

The core methodology developed in this thesis, of profiling the memory subsystem, identifying the bandwidth or latency bottlenecks, and designing a datapath to match the physical memory interface is not just limited to subgraph isomorphism, but rather this approach provides a useful template that can be applied to accelerate a wide range of other memory-bound algorithms, even from other fields using high-bandwidth FPGA platforms.

Bibliography

[1] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. «The ubiquity of large graphs and surprising challenges of graph processing: extended survey». In: *The VLDB journal* 29.2 (2020), pp. 595–618 (cit. on p. 1).

[2] Shixuan Sun and Qiong Luo. «In-memory subgraph matching: An in-depth study». In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1083–1098 (cit. on pp. 3, 4, 9, 12, 18).

[3] Roberto Bosio, Giovanni Brignone, Teodoro Urso, Mihai T Lazarescu, Luciano Lavagno, and Paolo Pasini. «Low-Power Subgraph Isomorphism at the Edge Using FPGAs». In: *IEEE Access* (2025) (cit. on pp. 5, 6, 15, 16, 18, 20).

[4] Alec Lu, Zhenman Fang, Weihua Liu, and Lesley Shannon. «Demystifying the memory system of modern datacenter FPGAs for software programmers through microbenchmarking». In: *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2021, pp. 105–115 (cit. on pp. 12, 13, 23, 24, 29).