Network-Integrated Decoding System for Real-Time Quantum Error Correction with Lattice Surgery

Namitha Liyanage, Yue Wu, Emmet Houghton, and Lin Zhong
Department of Computer Science, Yale University, New Haven, CT
Email : {namitha.liyanage, yue.wu, emmet.houghton, lin.zhong}@yale.edu

***Abstract*—Existing real-time decoders for surface codes are limited to isolated logical qubits and do not support logical operations involving multiple logical qubits. We present DECONET, a first-of-its-kind decoding system that scales to thousands of logical qubits and supports logical operations implemented through lattice surgery. DECONET organizes compute resources in a network-integrated hybrid tree-grid structure, which results in minimal latency increase and no throughput degradation as the system grows. Specifically, DECONET can be scaled to any arbitrary number of l logical qubits by increasing the compute resources by $O(l \times log(l))$, which provides the required $O(l)$ growth in I/O resources while incurring only an $O(log(l))$ increase in latency—a modest growth that is sufficient for thousands of logical qubits. Moreover, we analytically show that the scaling approach preserves throughput, keeping DECONET backlog-free for any number of logical qubits We report an exploratory prototype of DECONET, called DECONET/HELIOS, built with five VMK-180 FPGAs, that successfully decodes 100 logical qubits of distance five. For 100 logical qubits, under a phenomenological noise rate of 0.1%, the DECONET/HELIOS has an average latency of 2.40 μs and an inverse throughput of 0.84 μs per measurement round.**

I. INTRODUCTION

Quantum error correction (QEC) is essential for fault-tolerant quantum computing (FTQC). QEC employs multiple physical qubits to form a single, more resilient logical qubit, such as the surface code [1, 2]. A decoder identifies potential errors with these physical qubits before they lead to a logical error. *Lattice Surgery* [3] is a popular approach for implementing logical operations with logical qubits in surface codes. It merges and splits logical qubits, avoiding the need for non-local interactions but creating fresh challenges for the decoder design. See §II. Although significant progress has been made in decoder development [4, 5], all existing real-time decoder implementations are limited by the resources available from a single node (FPGA or computer). As a result, they support isolated logical qubits [6–13]. To the best of our knowledge, no real-time decoder implementation supports QEC with lattice surgery. See §III.

In this paper, we report DECONET, a scalable real-time quantum error correction system that overcomes resource limitations by utilizing multiple network-integrated FPGA nodes. DECONET distributes the decoding workload across several network-integrated compute resources, supporting larger code distances and multiple logical qubits. DECONET assigns a decoder instance for each logical qubit, which first decodes the logical qubit in isolation. The system then combines the results of the decoder instances to solve the global decoding problem.

DECONET organizes compute resources in a network of a hybrid tree-grid topology, allowing both computational power and I/O capacity to scale by adding FPGA nodes over the network. As a result, DECONET can decode large-scale circuits containing hundreds of logical qubits interacting through lattice surgery operations. Furthermore, DECONET can decode circuits with dynamic interactions, such as those involving conditional operations. No prior real-time, graph-based decoder can handle dynamic configurations, as they assume a static decoding graph (See §II-B). However, practical quantum algorithms involve conditional gates, resulting in a dynamic decoding graph that must be generated at runtime. DECONET efficiently decodes these dynamic graphs using a fusion operation.

We have prototyped DECONET using the Helios decoder [12] as decoder instances. The system, DECONET/HELIOS, consists of five Xilinx VMK-180 FPGA evaluation boards [14] arranged in a two-level tree topology. Additionally, DECONET/HELIOS features a custom low-latency interconnect for the four leaf nodes. We experimentally show that DECONET/HELIOS can decode up to 100 logical qubits at code distance 5, including realistic lattice surgery operations. For 100 logical qubits, DECONET/HELIOS achieves an average decoding latency of 2.40 μs and an inverse throughput of 839 ns per measurement round under 0.1% phenomenological noise. This result demonstrates, for the first time, the feasibility of parallel decoding for 100 interconnected logical qubits in real time.

In summary, we report the following contributions.

- The first scalable real-time QEC decoder capable of decoding thousands of logical qubits by distributing workloads across multiple network-integrated FPGA nodes, breaking the single-node resource constraint of previous decoders.
- The first real-time decoder capable of decoding lattice-surgery-based dynamic decoding graphs
- A set of empirical data demonstrating that DECONET, implemented using five FPGAs, can successfully decode 100 logical qubits with lattice surgery operations in real-time, a first for a decoder implementation.
- Fusion Union-Find, a novel method for parallelizing the decoding of large, dynamic decoding graphs, which is 25-40% faster than parallel window decoding.

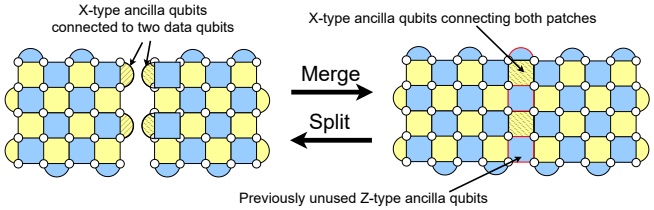


Fig. 1: Merging of two $d = 5$ surface code patches along their Z_L boundaries involves measuring joint ancillas across the boundary to form a single logical patch. Specifically, X-type ancilla qubits connected to two data qubits at the boundary (marked by arrows in the top left) are extended into ancilla qubits interacting with data qubits from both patches (arrows in the top right). Merging also activates previously unused Z-type ancilla qubits at the boundary, whose joint measurement yields the $Z_L Z_L$ operator. The newly activated Z-type ancillas in the merged qubit are shown with red borders.

II. Background

We provide brief overviews of the implementation of logical operations using lattice surgery, the requirements for decoders to support such operations, and the role of qubit controllers.

A. Lattice Surgery

Lattice surgery [3] is a resource-efficient method to implement logical operations with more than one logical qubits encoded in surface codes. As lattice surgery requires only nearest-neighbor physical qubit interactions, it enables logical circuits to be executed on planar quantum hardware without the need for long-range connections between qubits. This hardware-friendly property makes lattice surgery a promising candidate for scalable fault-tolerant quantum computing. As a result, researchers have embraced lattice surgery over alternative methods such as transversal gates [15–19].

Lattice surgery involves *merging* and *splitting* surface code patches to perform multi-qubit logical operations. A merge operation joins the adjacent boundaries of two surface code patches by measuring ancilla qubits that interact with data qubits from both logical qubits across the boundary, as illustrated in Figure 1. Depending on whether the X_L or Z_L boundary is merged, this process consists of two concurrent steps: (1) applying additional gates between data qubits at the boundary of one logical qubit and ancilla qubits at the boundary of the other, (2) measuring previously unused ancilla qubits at the boundary. In Figure 1, merging occurs along the Z_L boundary: it converts the X-type ancilla qubits at the boundary into ancilla qubits connecting both logical qubits, and measures previously unused Z-type ancillas (highlighted with red borders). This approach is generalizable to more than two logical qubits, and two distant logical qubits can be merged using a chain of ancillary logical qubits [3].

Splitting is the reverse operation of merging. After merging, the combined logical patch can be split back into two separate logical qubits by ceasing joint stabilizer measurements and resuming individual stabilizer measurements for each patch.

Lattice surgery operations, especially merge, pose additional challenges to QEC decoding. The merging of logical qubits results in shared error syndromes between the qubits. Since any two logical qubits in the system can potentially be merged, the worst-case scenario requires the decoder to handle a merged patch that spans the entire qubit array. This necessitates a decoder capable of processing all logical qubits collectively, as independent decoding of each logical qubit would fail to accurately decode the shared error syndromes introduced by merging.

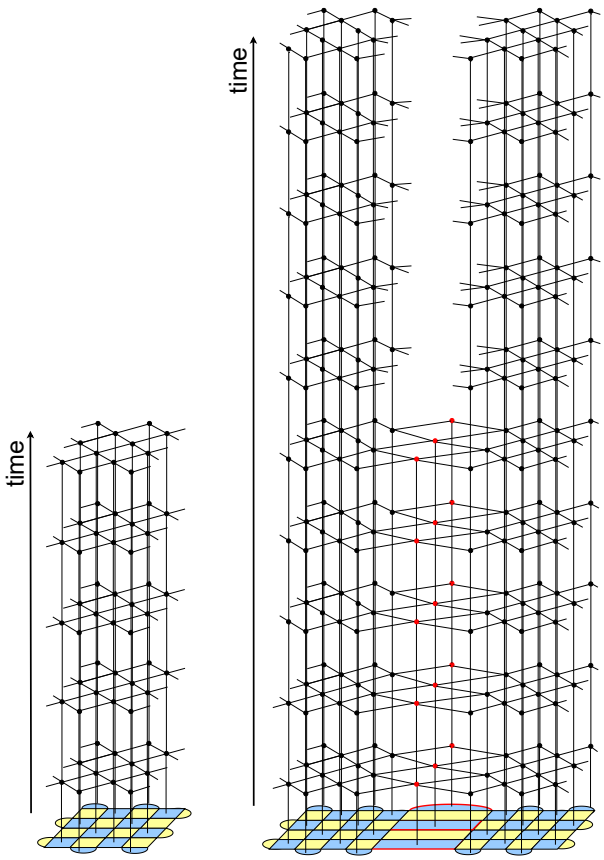


Fig. 2: Lattice surgery modifies the structure of the decoding graph, assuming under a phenomenological noise model for a $d = 5$ surface code. (top) Decoding graph of Z-ancillas over five measurement rounds. (bottom) Decoding graph of a system involving two logical qubits that are merged for five rounds and then split. The decoding graph contains additional vertices (marked in red) when the two logical qubits are merged.

In comparison, splitting is much simpler. Once a patch is split, two decoders can decode the resulting logical qubits independently. Decoders can exploit this parallelism to improve decoder throughput when managing multiple logical qubits.

B. QEC Decoders

When measurements of the ancilla qubits are ready, a decoder, using classical computing, identifies the most likely set of physical errors that could cause the observed defect measurements. To mitigate the effects of measurement errors, the decoder processes at least d rounds of measurements together, collectively referred to as a syndrome. With lattice surgery, the decoder must decode d rounds of measurements together after each merge or split operation [3].

Given the syndrome, the problem of finding the most likely errors is commonly represented by a *decoding graph*. Each vertex in the decoding graph corresponds to an ancilla measurement, and each edge represents a potential error. Figure 2 (top) illustrates the decoding graph for a single logical qubit coded in a surface code with $d = 5$. For example, Sparse Blossom [7] and Parity Blossom [20] solve this problem exactly, while Union-Find (UF) [21] solves this problem approximately [22] but faster.

1) Dynamic Nature of Decoding Graphs: With lattice surgery, the decoding graph becomes *dynamic*, because the graph structure depends on the outcomes of previous measurements and conditional operations in the circuit. For instance, whether two logical qubits should be merged may be determined by the result of a measurement on a third qubit, only available at runtime, as in the case of a T-gate implemented using magic state injection [23]. As a result, the complete decoding graph cannot be precomputed, and the decoder must construct the decoding graph as the computation progresses.

This runtime graph construction poses new challenges to decoding. In particular, the decoder must not only process syndrome data in real time, but also update the decoding graph itself based on conditional control flow. This requires tight integration with the control stack and minimal latency to keep up with real-time circuit execution. In Figure 2 (bottom), two logical qubits are merged for d rounds and then split apart. Whether this merge occurs can depend on a prior measurement result, meaning that the decoding graph, especially the inclusion of vertices corresponding to shared ancillas during the merge stage (shown in red), is not known ahead of time.

2) Performance Metrics: The performance of a decoder is evaluated using three metrics: accuracy, throughput, and latency. *Accuracy* determines the size of the surface codes to achieve a given logical error rate. Decoders with lower accuracy require much more hardware. For current physical error rates in superconducting quantum computers, $p = 0.001$, a Union-Find decoder requires $d \approx 29$ to achieve a logical error rate of 10^{-15}. *Latency* determines the rate of logical operations, as intermediate decoding results are often needed for time-sensitive operations such as T-gates [24]. As a result, latency determines the overall execution time of the circuit and the probability of failure. For superconducting quantum computers, researchers typically assume a decoding latency budget of 10 μs when estimating the timing of quantum algorithms [25]. *Throughput* or rate of decoding must match the measurement rate. Otherwise, a backlog of undecoded measurements can accumulate, exponentially slowing down the quantum computer. For superconducting quantum computing, which has the most stringent requirements, the decoding rate should be $1\mu s$ per measurement round [26].

In this context, a *real-time* decoder is one whose throughput exceeds the measurement rate and whose latency is comparable to the time to measure d rounds. For state-of-the-art superconducting quantum systems, this corresponds to a throughput greater than one million measurements per second and a total latency of approximately d μs [27].

C. Qubit Controller

In a quantum computer, the decoder must receive measurements from the qubit controllers. Using either an FPGA or specialized hardware, a controller generates the control signal to manipulate a qubit. A quantum computer needs 100s of qubits and therefore 100s of controllers in parallel to support experiments with many logical qubits [28–34].

This poses specific design requirements for the decoder. First, the decoder must be tightly integrated with qubit controllers to aggregate data from all the controllers with minimal latency. Second, the decoder's compute and I/O communication capabilities must scale with the number of qubit controllers to prevent it from becoming a system bottleneck.

III. Related work

To our knowledge, DECONET/HELIOS is the first scalable real-time decoder system that provides empirical results on decoding hundreds of interacting logical qubits simultaneously, including support for dynamic lattice surgery circuits. DECONET/HELIOS draws inspiration from several works that explored decoding lattice surgery-based dynamic graphs. It is the first decoder system that is capable of using network-integrated compute resources and supports real-time decoding with thousands of logical qubits.

Wu et al. [35] suggest a distributed decoding system in which a coordinator assigns decoding blocks to multiple compute resources and merges them using a fusion operation. DECONET/HELIOS can be considered the first implementation of this distributed decoding system, with new implementation ideas such as combining fusion with windowing to reduce latency and statically assigning decoding blocks to hardware units for more efficient execution. Bombin et al. [18] propose a modular decoding framework that partitions the decoding graph into subgraphs. These subgraphs correspond to commonly used logical operations. The framework first decodes the boundaries between these subgraphs, followed by parallel decoding of the individual subgraphs. The authors also considered dynamic circuits and proposed a hardware architecture in which multiple processing units, sharing a global memory, decode subgraphs in parallel, without providing a hardware implementation or any empirical data on decoding latency or throughput.

Lin et al. [17], Skoric et al. [36], and Tan et al. [37] propose partitioning the decoding graph into spatial regions, or windows, that can be decoded in parallel with limited inter-window dependencies. Lin et al. further analytically show that inter-window communication latency does not impose a scalability bottleneck. Our work builds on both these efforts to implement window-based decoding across FPGAs and, for the first time, *empirically* validating that inter-FPGA communication is not a scalability bottleneck.

To our knowledge, QULATIS [19] is the only reported hardware *design* that supports lattice surgery-based dynamic decoding graphs. It decodes errors using a greedy algorithm

that sacrifices accuracy for simplicity. Without an implementation, the authors evaluated the decoding latency of QUALTIS using SPICE simulations. Compared to QUALTIS, DECONET/HELIOS features a much more scalable architecture that leverages network-integrated FPGAs. Furthermore, our implementation of DECONET/HELIOS employs a Union-Find-based decoder, achieving at least two orders of magnitude higher decoding accuracy compared to the greedy algorithm used in QULATIS.

IV. DECONET OVERVIEW

A. Design Goals

Our goal is to build a system capable of real-time decoding of multiple interacting logical qubits, implemented by the surface code. In graph-based decoding, this means handling a dynamic decoding graph comprising all logical qubits in the quantum computer. The system should be both scalable and adaptable. *Scalability* refers to the system's capability to handle increasingly larger decoding graphs by incrementally and trivially adding compute resources without significant redesign. *Adaptability* refers to the system's ability to support a dynamic decoding graph whose complete structure is not fully known before runtime.

B. Key Ideas

DECONET achieves its design goals by integrating several key ideas:

Partitioning the decoding graph into decoding blocks: Since a monolithic decoder cannot scale efficiently due to hardware resource constraints, we partition the decoding graph into smaller units independently decodable, called decoding blocks, as inspired by [35]. The system partially decodes each decoding block separately using a decoder and combines them to form a global decoding solution.

Network-Integrated Resources for Scalability: DECONET uses network-integrated compute resources to scale beyond the limitations of a single node, computer, or FPGA. Unlike board- or chip-level integration, network integration allows us to incrementally and trivially add resources to scale the design.

Fusion-based and Window-based Approaches: We use two complementary methods to combine decoding blocks into a global decoding solution. Within a single computational resource (such as a single FPGA), we use a fusion-based approach [20, 35]. Across multiple network-integrated compute resources, we adopt a window-based approach [17]. We detail these methods and our partitioning strategy in §V.

Hybrid Tree-Grid Network Topology: To minimize communication latency among compute resources, we organize the compute resources in a novel hybrid topology which we describe in detail in §VI. The hybrid topology uses a grid structure to facilitate local communication required for lattice surgery operations among neighboring compute resources, and a tree structure to ensure minimal worst-case latency when communicating decoding outcomes between compute resources. Additionally, the hybrid topology allows I/O resources to scale proportionally with the number of compute resources.

The design of DECONET is agnostic to the choice of decoder implementation. This flexibility is crucial as decoding technologies rapidly evolve, each offering different trade-offs in accuracy, latency, and resource utilization. DECONET only poses two requirements to the decoder: first, it must decode a decoding block of $\approx d^3$ vertices in real-time; and second, it must support the fusion of decoding blocks across both space and time. These requirements are reasonable in view of recent advancements in decoder designs. Many decoders meet the real-time requirement for moderate values of d [6–13, 20, 27]. Several recent decoders also support fusion [13, 20]; both UF and MWPM decoders can be extended to support fusion.

We implement DECONET using Helios as decoder instances, referred to as DECONET/HELIOS (detailed in §VII). This implementation spans five FPGAs and can decode up to 100 logical qubits at $d = 5$.

V. DECONET DECODING ALGORITHM

This section describes the decoding algorithm used in DECONET. It outlines how DECONET partitions the decoding graph into decoding blocks, distributes them across compute resources to decode them in parallel, and combines them to form the complete graph.

A. Decoding Block

A decoding block is the basic unit of decoding in DECONET. For a logical qubit of code distance d, a decoding block consists of d rounds of ancilla measurement outcomes. As FTQC requires decoding at least d rounds of measurements after a logical operation, a decoding block is the smallest subgraph in the decoding graph capable of producing a meaningful result.

Dividing the decoding graph into decoding blocks enables fast construction of a dynamic decoding graph. Each decoding block has a static graph structure that DECONET generates offline. At runtime, DECONET constructs the dynamic decoding graph from these pre-constructed blocks. Thus DECONET represents graph changes such as merges and splits between logical qubits as merges and splits between decoding blocks. By updating only the connections between affected decoding blocks, DECONET avoids reconstructing the entire decoding graph at runtime, which would be time-intensive.

Decoding blocks also increases parallelism. When logical qubits are split, DECONET decodes their decoding blocks in parallel. DECONET combines relevant blocks only when logical qubits merge, after initially decoding them in parallel.

B. Combining Decoding Blocks

We consider two scenarios when combining decoding blocks: (1) combining within a single compute resource and (2) combining across network-integrated compute resources.

1) Intra-resource Decoding Block Fusion: DECONET uses a fusion operation previously suggested by Wu et.al. [20, 35] to merge blocks within the same compute resource both spatially and temporally. Fusion provides few advantages compared

to alternate combining methods such as parallel window decoding and sliding window decoding.

- *Reduced Redundant Computation:* Conventional methods require overlapping windows of at least $d/2$ width along every direction to combine blocks, causing redundant computations in the overlapping region. Furthermore, the redundant computation region scales $O(d^3)$. In contrast, when using a clustering-based decoding algorithm, fusion avoids redundant computations as it preserves the cluster details from the first decoding stage when starting the fusion operation. Furthermore, fusion only affects clusters incident to the boundary between merged blocks, which scales $O(d^2)$.
- *Increased Parallelism:* Conventional methods impose sequential dependencies due to artificial defects along boundaries [1, 17, 36]. Fusion eliminates this bottleneck by allowing simultaneous decoding of all the blocks prior to combining. This is especially advantageous when merging blocks along the time domain. When using fusion decoding, the first block can start after the first d rounds are available, whereas sliding window decoders must wait for all $2d$ rounds to start decoding.

2) Inter-resource Decoding Block Combination via Parallel Window Decoding: We use the parallel window decoding method, proposed by Lin et al. [17] and Skoric et al. [36], to combine decoding blocks across compute resources due to its lower communication overhead. This method reduces communication latency by requiring only one unidirectional message per pair of adjacent decoding blocks, unlike fusion-based methods that involve multiple data exchanges. This reduction is critical when combining decoding blocks across the network, where communication latency often exceeds decoding latency. For example, state-of-the-art FPGA decoders can decode a d=21 surface code in 240 ns [38], but communication between FPGAs over gigabit transceivers can take around 300 ns [39].

The parallel window decoding method partitions the decoding graph into multiple windows [17, 36]. In the context of DECONET, each window consists of decoding blocks mapped to the same compute resource. Parallel window decoding groups these windows into multiple groups such that no two adjacent windows belong to the same group. This method then decodes the groups sequentially, performing parallel decoding of windows within each group and transmitting boundary information between groups in order. This boundary information represents the most probable physical qubit errors that cross window boundaries.

Prior work shows that at least three groups are required to satisfy the adjacency constraint [17, 36]. Because the number of groups directly impacts decoding latency, DECONET groups the network-integrated compute resources into three groups to minimize latency while satisfying the constraint.

C. Pipelined Decoding Procedure

Next, we describe the general decoding procedure of DECONET that organizes the compute resources into three groups as motivated above. Figure 3 illustrates the pipelined execution of this procedure across the three groups.

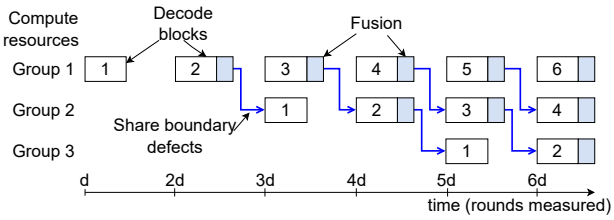


Fig. 3: Timeline of the decoding procedure pipelined across three compute resource groups. Each group performs parallel decoding of its assigned decoding blocks (white boxes). After decoding, DECONET fuses adjacent decoding blocks in space and time (blue). The system then shares the boundary information between groups (blue arrows). Numbers in the white boxes denote the round index of decoding blocks. Groups 2 and 3 lag behind group 1 due to data dependencies.

When the $d \cdot i^{th}$ round of measurements becomes available, Group 1 decodes the corresponding decoding blocks assuming they are isolated. After local decoding, Group 1 fuses adjacent decoding blocks corresponding to merged logical qubits and performs time-domain fusion with decoding blocks from the $(i-1)^{th}$ round corresponding to the same logical qubit. Once fusion completes, Group 1 commits the $(i-1)^{th}$ round and transmits boundary information to Group 2. Group 2 then decodes and fuses the $(i-1)^{th}$ round using this information and commits the $(i-2)^{th}$ round. Similarly, Group 3 processes and commits the $(i-3)^{th}$ round.

This pipelined execution introduces a minimum decoding latency of $3d$ rounds but does not affect the throughput of decoding. Each group begins decoding a new round of decoding blocks immediately after it commits the previous round and receives the required boundary information.

VI. DECONET NETWORK ARCHITECTURE

The key to DECONET's scalability is to exploit network-integrated compute resources, allowing it to go beyond a single compute node, e.g., FPGA, which has limited prior decoder implementations such as Helios [38], Lilliput [11], and Micro Blossom [13]. We next describe the network architecture of DECONET, which organizes the compute resources in a hybrid tree-grid topology for scalable and efficient decoding.

A. Hybrid Tree-Grid Network Topology

Figure 4 illustrates the hybrid tree-grid network topology employed by DECONET. Compute resources at the lowest level (leaf nodes) run decoder instances, while intermediate nodes act as routers. The root node serves as the central interface for user interaction, configuration, and experiment monitoring.

This hybrid tree-grid topology brings the benefits of both tree and grid. The *grid structure* efficiently handles local communication between adjacent leaf nodes. Through careful mapping of decoding blocks to leaf nodes (§VI-C), DECONET restricts boundary information exchange exclusively to adjacent compute resources in the grid. By restricting boundary

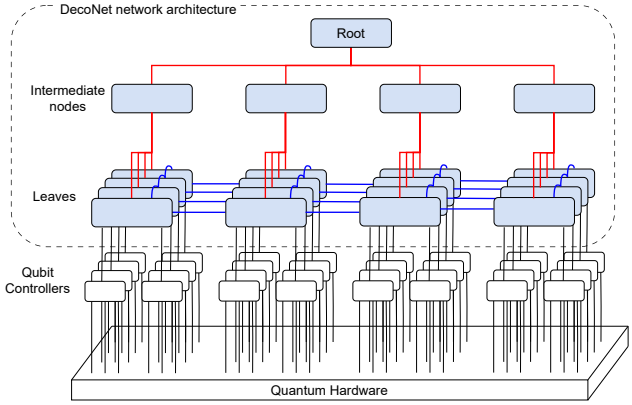


Fig. 4: Network Architecture of DECONET, showing the hybrid tree-grid structure. The tree's leaf nodes run decoder instances, while intermediary nodes route information.

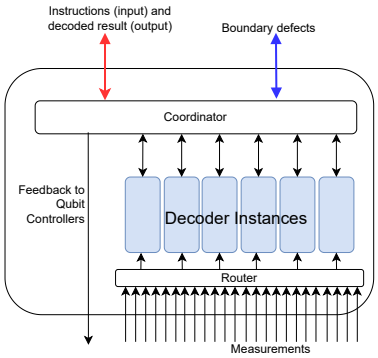


Fig. 5: Internal architecture of DECONET leaf nodes, comprising a coordinator, router, and multiple decoder instances.

information to point-to-point grid connections, we eliminate routing overheads.

However, a purely grid-based structure becomes inefficient under worst-case communication scenarios, such as transferring logical states between distant nodes. Grid-only topologies result in a worst-case latency scaling of $O(\sqrt{n})$, where n denotes the number of logical qubits. To mitigate this, we augment the grid with a *tree structure*, reducing maximum communication latency between any two nodes to $O(\log(n))$. The tree structure handles both time-sensitive (logical results, control signals) and non-time-sensitive data (initial decoder configurations, monitoring signals).

We could not find a similar hybrid tree-grid topology in the literature, likely due to quantum computing having unique requirements: strong locality of information sharing for lattice surgery, where a grid topology excels, and the need to minimize worst-case latency, where a tree topology is ideal.

Furthermore, this topology supports scalable expansion of I/O resources. By directly connecting qubit controllers to leaf nodes, the system can scale its support for additional qubit controllers by incorporating more leaf nodes and, if necessary, adding intermediate routing nodes to expand the tree. Additionally, the same grid network can also facilitate the distribution of clock signals to qubit controllers.

To minimize worst-case latency, maintaining a minimal height for the DECONET tree structure is beneficial. The branching factor at each node and the necessary I/O capacity for qubit controllers, both dependent on the compute resource's maximum fan-out, dictate the tree's height. We leave the specific tree configuration as an implementation choice.

B. Network Components of DECONET

1) Root Node: The root node consists of three components: a user interface, a logical-level processor, and a router. The user interface facilitates system configuration, status monitoring, and end-user interaction, such as providing an input logical circuit. The logical-level processor executes the logical quantum circuit provided by the user and generates decoding instructions based on this circuit. The router distributes these instructions to leaf nodes and forwards messages from the leaf nodes to the logical-level processor.

2) Leaf Nodes: Figure 5 shows the internal architecture of a leaf node, which comprises a coordinator, a router, and one or more decoder instances. The coordinator controls the decoding workflow, interpreting and executing instructions from the root. This includes configuring boundaries between decoding blocks, initiating decoding operations on decoder instances, and communicating boundary information between leaf nodes. We statically configure the router to route measurement data to the appropriate decoder instances. The decoder instances decode the blocks and combine them according to the boundaries between blocks.

3) Intermediate Nodes: Intermediate nodes route messages between the root node and leaf nodes.

C. Mapping Decoder Instances to Compute Resources

We need to efficiently map decoding blocks to leaf nodes to minimize inter-resource communication. Our strategy of mapping is as follows.

We statically assign each decoder instance to decode blocks corresponding to one logical qubit. This provides two benefits. First, combining decoding blocks across the time domain does not require inter-resource communication. Second, by connecting the relevant qubit controllers directly to this leaf node, we reduce measurement transmission latency, which contributes to the overall decoding latency.

We spatially group adjacent decoding blocks to the same node whenever possible, minimizing communication across the network. When we must spread adjacent decoding blocks across the network to multiple nodes due to resource constraints, we map them to compute resources that can directly communicate via the grid network, thereby localizing traffic and reducing overall network congestion.

VII. DECONET/HELIOS IMPLEMENTATION

We built a concrete example of DECONET, called DECONET/HELIOS, using five VMK180 evaluation boards, each

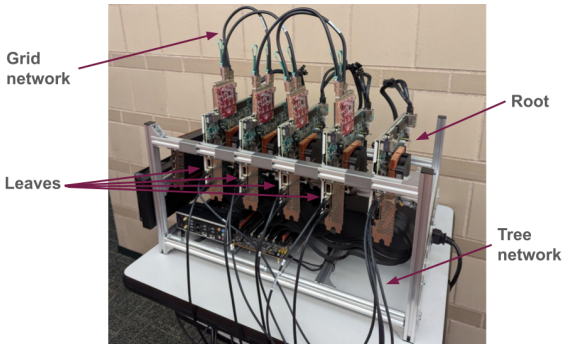


Fig. 6: Implementation of DECONET/HELIOS using five FPGAs. The rightmost FPGA is the root of the tree, and the other four FPGAs are leaf nodes.

featuring a Versal VM1802 FPGA-based SoC [14]. We chose these boards because they are one of the readily available Versal FPGA development boards with a very high LUT count and a high number of gigabit transceivers. Figure 6 shows the implementation DECONET/HELIOS. The network of our implementation is a two-level tree, with one root node and four leaf nodes, which we also connect in a grid. We next describe the important choices made in our implementation.

A. Network Connection between FPGAs

We implemented the FPGA interconnection using Gigabit Transceivers to maximize fan-out. Each VMK180 board includes 30 GTs exposed through QSFP, SFP+, and FMC+ links, allowing each parent node to connect to up to 25 child nodes. This high fan-out capability significantly reduces DECONET's tree height, enhancing scalability when the system grows.

The Aurora core, the standard Xilinx IP core designed for high-throughput communication [39], incurs a high core-to-core latency of approximately 320 ns due to its optimization for throughput, making it unsuitable for real-time decoding. To address this limitation, we developed a custom low-latency transceiver, the Eos core [40], which reduces latency to 95 ns by sacrificing throughput. Eos core breaks messages into smaller chunks and inserts more frequent error correction codes, allowing the transmitter and receiver to process data in fewer cycles, minimizing overall latency. Despite the trade-off in throughput, the Eos core supports stable operation at 16 Gbps, which is sufficient for real-time decoding. We validated its reliability through 24-hour stress testing, confirming its suitability for DECONET. The Eos core is open-source and available at [40].

1) Messaging format: We use a fixed 64-bit format for messages exchanged between FPGAs. The first 8 bits specify the destination FPGA, the next 8 bits define the message header, and the remaining 48 bits serve as the payload. This simple format enables faster routing at each node, which is necessary for low-latency communication.

B. Choice of Decoder Instance: Helios

We chose the Helios decoder as the decoder instance due to its favorable trade-off between scalability, latency, and accuracy [12]. Helios is a distributed implementation of the Union-Find (UF) decoder [21], which offers slightly lower accuracy than minimum-weight perfect matching (MWPM) decoders such as those used in Micro-Blossom [13]. However, Helios achieves significantly lower latency and can decode surface codes up to $d = 21$ in under 250 ns. This low latency enables us to demonstrate that DECONET can support low-latency decoding.

Since Helios is based on the Union-Find (UF) algorithm, we extended it to support the fusion operation, referred to as Fusion Union-Find (Fusion UF). We provide implementation details in §VII-E.

The choice of decoder instance presents a design trade-off. More accurate but resource-intensive decoders such as Micro-Blossom [13] can improve logical error rates but increase decoding latency and reduce the number of logical qubits that can be supported per FPGA. In contrast, more resource-efficient UF-based decoders such as LCD [27] can support more logical qubits per FPGA but incur higher decoding latency. Helios strikes a balance by offering low latency and moderate resource usage while maintaining acceptable decoding accuracy, making it suitable for our implementation.

C. Implementation of FPGA logic

We implemented the FPGA logic using Verilog and Tcl scripts, comprising approximately 9000 lines of code. The source code is publicly available at [41]. To support multi-qubit decoding, where boundaries dynamically change, we maintain an array of registers to track the state of each boundary. The coordinator has write access to these registers and updates them based on the instructions from the root. Each Helios instance within the FPGA has read access to these registers, allowing it to determine the current state of the boundary for the logical qubit it is decoding.

D. Resource Usage

Table I presents the resource usage breakdown for the DECONET/HELIOS implementation configured to decode 100 logical qubits at $d = 5$. This configuration requires close to 1 million LUTs distributed across five FPGAs.

In the leaf nodes, decoder instances consume approximately 95% of the LUTs in the implementation, which is expected since decoding is the most compute-intensive task in the system. Adopting more resource-efficient decoder instances, such as [42], could potentially increase the system's decoding capacity per FPGA. We further explore these scalability limitations in §VIII-D.

In contrast to the decoder instances, the remaining logic in the leaf nodes consumes only around 14,000 LUTs and 21,000 registers, accounting for approximately 1.5% of the FPGA's resources. This small footprint leaves sufficient room for the compute-intensive decoder instances. The 1.5% includes the coordinator logic, inter-FPGA communication links, block-RAM-based FIFOs for inter-module communication, and peripheral logic for interacting with the ARM core.

The root node utilizes only 2% of the resources on the evaluation board, making it possible to use FPGAs with

TABLE I: Breakdown of resource usage for the configuration decoding maximum number of logical qubits, 100 ($d = 5$)

Component	LUTs	Registers	BRAMs
Root Node			
Eos core	4389	5365	0
Root Coordinator	53	646	0
Residual Logic	14004	8274	46
Leaf Node (per FPGA)			
Coordinator	25,038	3,859	0
Decoder Instances	201,022	98,981	0
Eos core	6141	7317	0
Residual Logic	7753	13542	40

lower LUT counts for non-leaf nodes in DECONET. However, existing evaluation boards with smaller LUT capacities typically have fewer transceiver links, preventing us from pursuing this option. Using a board with fewer transceiver links would increase the tree height, leading to higher decoding latency. Alternatively, the unused resources in non-leaf nodes can potentially be repurposed for other tasks in the quantum control stack, such as logical qubit routing.

E. Fusion Union-Find

We introduce Fusion Union-Find (Fusion UF), our novel approach for merging multiple partitions of a decoding graph, decoded using the Union-Find algorithm. This is an alternative to conventional merging techniques such as sliding window decoding and parallel window decoding [36, 37]. Fusion UF draws inspiration from Fusion Blossom [20], which uses a similar methodology to speed up MWPM-based decoding.

1) Merging using Fusion Union-Find: Fusion UF merges two blocks as follows. Initially, the Distributed Union-Find decoder processes the two blocks independently, treating their shared face as an artificial boundary. Any cluster with fully grown edges that touch this artificial boundary is considered even and do not grow further.

After completing the clustering phase in both blocks, the system removes this artificial boundary and calculates the cluster parities again. If any cluster in either block is odd, the decoder resumes the growing and merging phase in both blocks simultaneously until no more odd clusters remain. Finally, the system moves to peeling phase in each block.

In §VIII-C, we present empirical evidence demonstrating that Fusion UF achieves lower latency compared to other methods.

VIII. EVALUATION

The main objective of our evaluation is to assess the scalability of DECONET. To that end, we answer the following key questions:

- **Latency and throughput growth**: How do latency and throughput scale with the code distance and the number of merged logical qubits?
- **Limitations**: What are the scalability limits of DECONET?
- **Accuracy**: How does the fusion-UF approach compare in accuracy to alternative methods?

We first describe our methodology and then present empirical results answering these questions. To validate the system's practicality, we decode a set of micro benchmarks.

A. Methodology

To evaluate the accuracy of fusion-UF, we extend the software simulation library [43] to support fusion-UF. Software simulations enable us to test higher code distances (d) without encountering hardware resource constraints.

For latency and throughput evaluations, we measure FPGA clock cycles required for syndrome decoding using DECONET/HELIOS. We define latency as the time between the availability of the last measurement round for a decoding block at the decoder and the availability of the decoded result. Inverse throughput is the time interval between a decoder instance accepting consecutive decoding blocks, normalized by d. We analyze the trends in latency and throughput to identify system limitations.

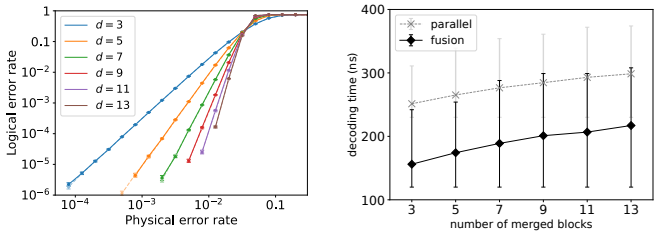
1) Experimental setup: We use the five-FPGA Helios-Net implementation described in §VII as the experimental platform. The ARM cores on each evaluation board generate sample syndromes and transfer them to the decoder instances in the FPGAs. To verify the correctness of our multi-FPGA implementation, we compare the logical state of each decoding block after every decoding round with results from offline simulations of the original UF decoder by Delfosse et al. [21], executed on the ARM core. The comparison reveals identical results between the software simulation and the hardware implementation. We perform 10^6 trials for each error rate and code distance.

We use two configurations of the Helios decoder as decoder instances in our evaluation. In most experiments, we use the default Helios (Helios-1) configuration, which offers the best latency scalability with d. To demonstrate the decoding of the maximum number of logical qubits, we use Helios-n, where $n = d$. Helios-n requires fewer FPGA resources than Helios-1, supporting more decoding blocks on the implementation.

We use $d = 5$ as our default configuration, as we believe it would be a reasonable distance for the first experiments with multiple logical qubits on actual quantum hardware.

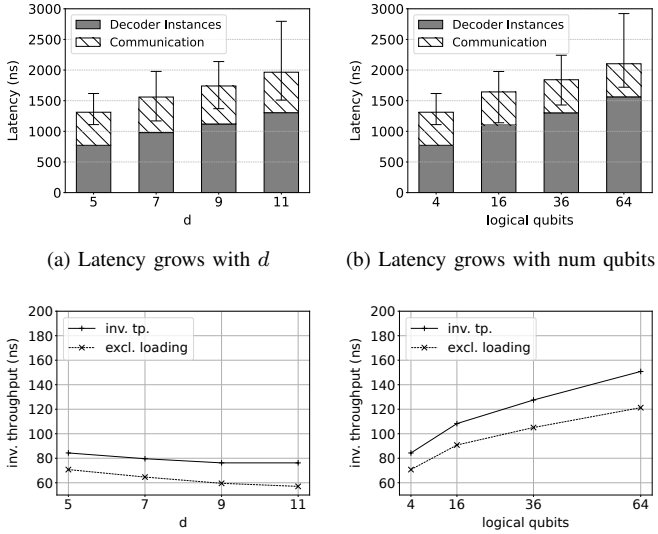
2) Noise Model: We use the phenomenological noise model [1] with measurement errors for our experiments. Prior studies widely use this model for single-qubit decoders [6, 38, 44]. The system can be easily extended to other noise models, such as circuit-level noise and erasure errors, as the Helios decoder supports both models.

For most of our experiments, we use a default noise level of $p = 0.001$, consistent with prior works [6, 38, 42]. This is a reasonable assumption as $p = 0.001$ is more than 10 times below the threshold, which is necessary to exponentially reduce the logical error rates. Furthermore, for scalability evaluations, we randomly merge and split adjacent decoding blocks with a 50% probability after every d rounds.



(a) No notable accuracy loss (b) Lower latency

Fig. 7: Accuracy and scalability of the fusion-UF approach. (a) Comparison of decoding accuracy for two merged blocks using fusion-UF and a global UF decoder. Solid lines correspond to results from fusion-UF, and dashed lines represent results from a global decoder. Each data point aggregates 10^8 trials. (b) Latency growth of fusion-UF compared to parallel window decoding for multiple merged logical qubits of $d = 5$. The error bars indicate the variation of latency from the minimum to the 95^{th} percentile.



(a) Latency grows with d (b) Latency grows with num qubits

(c) I reduces with d (d) I grows with num qubits

Fig. 8: Scalability of DECONET/HELIOS in terms of latency and inverse throughput (I). In (a) and (b), we plot the overall decoding latency as a function of code distance d and the number of logical qubits per FPGA. We break down the total latency into contributions from decoder instances and inter-FPGA communication. In (c) and (d), we show the inverse throughput (solid line) versus d and the number of logical qubits. The error bars indicate the variation of total latency from the minimum to the 95^{th} percentile. We plot the inverse throughput, with and without defect data loading time (solid and dashed lines, respectively). (a) Latency increases with code distance. (b) Latency increases with the number of logical qubits per FPGA. (c) Inverse throughput remains at least $10\times$ faster than the measurement rate for all distances. (d) Inverse throughput increases with the number of logical qubits but remains $6\times$ faster than the measurement rate even when decoding 64 logical qubits. Default parameters: $d = 5$, $p = 0.001$, and 4 logical qubits.

3) Micro Benchmarks: To validate the capabilities of our implementation, we create a set of microbenchmarks consisting of commonly required logical operations described in the literature that DECONET can decode.

B. Accuracy of Fusion-UF

To evaluate the accuracy of the fusion-UF approach, we decode two merged blocks with varying d using fusion-UF and a global UF decoder, which considers both blocks as a single decoding graph. Figure 7a illustrates the results. The results indicate that fusion-UF exhibits no statistically significant accuracy degradation, with a relative difference of less than 10% compared to the global-UF decoder. Prior work on parallel window and sliding window decoding also reports no noticable accuracy loss [17, 36].

C. Latency of Fusion-UF

We evaluate the scalability of Fusion-UF by comparing its decoding latency against parallel window decoding across varying numbers of merged logical qubits. Figure 7b shows the results, with the x-axis representing the number of merged logical qubits and the y-axis showing the decoding latency. Fusion-UF achieves lower latency than parallel window decoding across all tested configurations. The reduction of latency is primarily due to increased parallelism and reduced redundant computation in Fusion-UF as mentioned in §V-B.

D. Scalability of DECONET

We evaluate the scalability of the DECONET using its implementation DECONET/HELIOS with respect to d and the number of logical qubits. We use two primary metrics: latency and inverse throughput.

Figure 8a and Figure 8c show how latency and inverse throughput scale with increasing d, when each leaf FPGA decodes one logical qubit. Across all supported code distances, the overall latency remains below the d μs measurement interval typical of superconducting quantum systems. As shown in Figure 8a, latency consists of decoding within the decoder instances and inter-FPGA communication. While decoder execution scales sublinearly with d, inter-FPGA communication scales as $O(d^2)$, since it depends on the boundary size between decoding blocks. However, this scaling trend is not prominent in our results due to a 400 ns lower-bound latency imposed by the cumulative delay across network links.

The inverse throughput improves slightly from 84 ns to 76 ns as d increases from 5 to 11, consistent with trends observed in the Helios decoder for isolated qubits [12]. These values remain significantly below the 1 μs measurement interval, confirming that DECONET/HELIOS supports real-time decoding. Throughput is affected only by data loading and execution within a single decoding block, and remains independent of inter-FPGA communication latency due to the system's ability to pipeline loading of the next decoding block while transmitting boundary information.

The current implementation loads defects sequentially from the ARM core. However, in practice, the defects would arrive from multiple qubit control devices, enabling parallel loading up to the fan-out of each leaf node. When we exclude data loading time from our measurements, the inverse throughput decreases from 71 ns to 57 ns as d increases from 5 to 13.

Figure 8b and Figure 8d show how latency and throughput scale with the number of logical qubits. Latency increases from 771 ns to 1562 ns as the number of logical qubits grows from 4 to 64. This latency remains under 20% of the d μs measurement time. The inverse throughput increases from 71

TABLE II: Microbenchmark results showing latency and inverse throughput (standard deviation in brackets). Inverse throughput is normalized by d.

Microbenchmark	# L. Qubits	# Rounds	Latency (ns)	Inv. Thpt (ns)
Meas.-based feedback	1	d	916 (191.1)	86.9 (22.2)
Merge + split	2	$3d$	2003 (355.0)	84.5 (35.1)
Move qubit	3	$3d$	2087 (238.9)	87.2 (22.4)
CNOT	3	$3d$	3258 (619.9)	86.5 (24.6)
CNOT (plane layout)	6	$3d$	3351 (484.8)	91.5 (19.8)
Single-ctrl multi-CNOT	5	$3d$	3249 (482.7)	91.7 (34.8)
State expansion	4	$2d$	2751 (536.4)	86.4 (19.7)
15-1 magic state distill.	24	$5d$	5701 (633.0)	123.1 (35.9)

ns to 121 ns, which remains well below the 1 μs threshold.

Based on the experimental results showing that the inverse throughput scales sublinearly with d, the largest decoding block that can fit in an FPGA determines the maximum d. On VMK-180 SoCs, we support up to $d = 13$. However, the design could potentially scale up to $d = 23$ when using a VU19P FPGA, the largest commercially available option. Even though the number of decoding blocks per FPGA is also limited by logic utilization, this bottleneck is easily addressed by distributing blocks across additional leaf nodes.

The number of leaves and inter-FPGA latency do not impose immediate limits on scalability, as they impact latency but not throughput. However, increasing inter-FPGA latency reduces the logical operation frequency, leading to a polynomial increase in circuit execution time. Additionally, when the number of leaves increase, the system will eventually bottleneck at the root node, which determines circuit control paths based on prior decoding results. We can potentially avoid this bottleneck by taking distributed control decisions at intermediate nodes, but this direction requires further investigation.

E. Decoding 100 logical qubits

We decode up to 100 logical qubits of $d = 5$ using our five-FPGA implementation by employing the resource-efficient Helios-d configuration as the decoder instance. To support 100 logical qubits, each FPGA processes 25 logical qubits. Scaling beyond this point causes the decoding rate to fall below the measurement rate due to the limited scalability of the Helios-d configuration [12]. When decoding 100 logical qubits, the system achieves an average latency of 12.01 μs and an inverse throughput of 0.84 μs. While these values are significantly higher than those of the Helios-1 configuration, they remain within the bounds required for real-time decoding.

In terms of scalability, the Helios-d configuration reaches the decoding time limit before exhausting FPGA resources. Based on latency growth trends, we estimate that for an error rate of $p = 0.001$, Helios-d can support up to $d = 25$.

F. Microbenchmarks

We report the decoding latency and inverse throughput of selected micro benchmarks in Table II. Each entry lists the number of logical qubits and the number of measurement rounds required to implement the circuit and achieve fault-tolerant decoding. Because DECONET organizes decoding across FPGAs in a pipelined manner (Figure 3), some FPGAs process additional measurement rounds to complete the decoding of a logical circuit.

All microbenchmarks exhibit an average decoding latency below 30% of the measurement acquisition time. The inverse throughput is 8× faster than the measurement rate of 1 μs, ensuring that DECONET/HELIOS operates backlog-free across all benchmarks. We highlight two microbenchmarks that demonstrate key capabilities of DECONET: measurement-based feedback and 15-1 magic state distillation.

In measurement-based feedback, the system decodes a logical qubit and transmits the result to another FPGA, a control flow required in the T-gate. The quantum hardware must idle until the result is available, and circuit execution can slow polynomially due to this latency [36]. In our experiments, the average latency for decoding and transmitting the result to another FPGA is 0.91 μs, more than five times faster than the time to measure d rounds. Across 10^6 trials, we observe a worst-case latency of 2.25 μs, which is still over twice as fast as the time to measure d rounds. As DECONET/HELIOS can consistently deliver results before the next measurement round becomes available, it introduces minimal slowdown to quantum circuit execution.

15-1 magic state distillation, spanning 24 logical qubits, represents one of the largest merge-and-split-based operations required for FTQC. This circuit is essential for generating T-gates and involves five rounds of CNOT gates [45]. A practical decoder must process this circuit faster than the rate of measurement to be useful for large-scale FTQC. DECONET decodes the corresponding $5d$ rounds in 5.7 μs for $d = 5$, achieving inverse throughput 8× faster than the rate of measurement.

The decoding latency of each microbenchmark also depends on how many FPGAs participate in decoding, which is determined by the placement of the logical circuit. To evaluate worst-case behavior, we map each circuit across FPGA boundaries whenever possible, maximizing inter-FPGA communication. Mapping a circuit to a single FPGA reduces latency. For example, a CNOT gate requires 1297 ns on a single FPGA versus 3258 ns across three. However, inverse throughput remains statistically unchanged due to DECONET 's pipelined execution across all FPGAs.

G. Comparison with related work

We compare DECONET/HELIOS with QULATIS [19], the only prior hardware decoder design in the literature that provides detailed decoding latencies.

DECONET/HELIOS achieves significantly higher decoding accuracy than QULATIS, while QULATIS outperforms DECONET/HELIOS in latency and throughput. At $p = 0.001$ and $d = 5$, DECONET/HELIOS provides over two orders of magnitude better accuracy than QULATIS, and this gap widens with increasing d. This is due to QULATIS using a greedy decoding algorithm, which has orders of magnitude lower accuracy than Union-Find. QULATIS achieves lower latency due to its higher operating frequency. For 15-1 magic

state distillation at $d = 5$ and $p = 0.001$, QULATIS reports a latency of 1.16 μs based on SPICE-level simulation at 2 GHz, whereas DECONET/HELIOS achieves 5.7 μs when running at 100 MHz. In terms of cycle count, QULATIS requires 2235 cycles, while DECONET/HELIOS completes decoding in 570 cycles. For the same circuit, QULATIS achieves an inverse throughput of 46.7 ns, compared to 123.1 ns for DECONET/HELIOS, again primarily due to the clock frequency difference.

QULATIS also faces more stringent scalability limits. Its power consumption at cryogenic temperatures (4K) limits the number of supported logical qubits. Their estimation suggests it can run 40 concurrent 15-1 distillation circuits of $d = 9$. In contrast, DECONET 's scalability is limited by the root node's capacity to process decoded results, a bottleneck that does not emerge until scaling to thousands of logical qubits.

IX. CONCLUSION

We present DECONET, a network-integrated decoding system for fault-tolerant quantum computers that decodes a dynamic graph of multiple interacting logical qubits. DECONET introduces a scalable architecture that expands compute and I/O resources by trivially adding hardware, enabling the decoding of thousands of logical qubits. Using a five-FPGA implementation, DECONET/HELIOS decodes 100 logical qubits of distance five in real-time. To the best of our knowledge, this is the highest number of interacting logical qubits decoded faster than the measurement rate by any system. Given this scalability, we consider DECONET a strong candidate for the logical qubit decoding layer in future fault-tolerant quantum computers.

ACKNOWLEDGMENT

This work was supported in part by Yale University and NSF MRI Award #2216030.

REFERENCES

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, 2002.

[2] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, 2012.

[3] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, 2012.

[4] F. Battistel, C. Chamberland, K. Johar, R. W. J. Overwater, F. Sebastiano, L. Skoric, Y. Ueno, and M. Usman, "Real-time decoding for fault-tolerant quantum computing: Progress, challenges and outlook," *Nano Futures*, 2023.

[5] E. Campbell, "A series of fast-paced advances in quantum error correction," *Nature Reviews Physics*, vol. 6, no. 3, 2024.

[6] P. Das, C. A. Pattison, S. Manne, D. M. Carmean, K. M. Svore, M. Qureshi, and N. Delfosse, "AFS: Accurate, fast, and scalable error-decoding for fault-tolerant quantum computers," in *Proc. IEEE Int. Symp. High-Performance Computer Architecture (HPCA)*, 2022.

[7] O. Higgott and C. Gidney, "Sparse Blossom: correcting a million errors per core second with minimum-weight matching," *arXiv preprint arXiv:2303.15933*, 2023.

[8] S. Vittal, P. Das, and M. Qureshi, "Astrea: Accurate quantum error-decoding via practical minimum-weight perfect-matching," in *Proc. ACM/IEEE Int. Symp. Computer Architecture (ISCA)*, 2023.

[9] B. Barber, K. M. Barnes, T. Bialas, O. Buğdaycı, E. T. Campbell, N. I. Gillespie, K. Johar, R. Rajan, A. W. Richardson, L. Skoric, C. Topal, M. L. Turner, and A. B. Ziad, "A real-time, scalable, fast and resource-efficient decoder for a quantum computer," *Nature Electronics*, vol. 8, no. 1, 2025.

[10] W. Liao, Y. Suzuki, T. Tanimoto, Y. Ueno, and Y. Tokunaga, "WIT-Greedy: Hardware system design of weighted iterative greedy decoder for surface code," in *Proc. ACM Asia & South Pacific Design Automation Conf. (ASPDAC)*, New York, NY, USA, 2023.

[11] P. Das, A. Locharla, and C. Jones, "LILLIPUT: a lightweight low-latency lookup-table decoder for near-term quantum error correction," in *Proc. ACM Int. Conf. Architectural Support for Programming Languages & Operating Systems (ASPLOS)*, 2022.

[12] N. Liyanage, Y. Wu, S. Tagare, and L. Zhong, "FPGA-based distributed union-find decoder for surface codes," *IEEE Trans. Quantum Engineering*, 2024.

[13] Y. Wu, N. Liyanage, and L. Zhong, "Micro Blossom: Accelerated minimum-weight perfect matching decoding for quantum error correction," in *Proc. ACM Int. Conf. Architectural Support for Programming Languages & Operating Systems (ASPLOS)*, 2025.

[14] Xilinx Inc., "VMK180 Evaluation Board," 2021. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vmk180.html

[15] L. Lao, B. v. Wee, I. Ashraf, J. v. Someren, N. Khammassi, K. Bertels, and C. G. Almudever, "Mapping of lattice surgery-based quantum circuits on surface code architectures," *Quantum Science and Technology*, vol. 4, no. 1, 2018.

[16] G. Watkins, H. M. Nguyen, K. Watkins, S. Pearce, H.-K. Lau, and A. Paler, "A high performance compiler for very large scale surface code computations," *Quantum*, vol. 8, 2024.

[17] S. F. Lin, E. C. Peterson, K. Sankar, and P. Sivarajah, "Spatially parallel decoding for multi-qubit lattice surgery," *arXiv preprint arXiv:2403.01353*, 2024.

[18] H. Bombín, C. Dawson, Y.-H. Liu, N. Nickerson, F. Pastawski, and S. Roberts, "Modular decoding: parallelizable real-time decoding for quantum computers," *arXiv preprint arXiv:2303.04846*, 2023.

[19] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, "QULATIS: A quantum error correction methodology toward lattice surgery," in *Proc. IEEE Int. Symp. High-Performance Computer Architecture (HPCA)*, 2022.

[20] Y. Wu and L. Zhong, "Fusion blossom: Fast MWPM decoders for QEC," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2023.

[21] N. Delfosse and N. H. Nickerson, "Almost-linear time decoding algorithm for topological codes," *Quantum*, 2021.

[22] Y. Wu, N. Liyanage, and L. Zhong, "An interpretation of union-find decoder on weighted graphs," *arXiv preprint arXiv:2211.03288*, 2022.

[23] D. Litinski, "A game of surface codes: Large-scale quantum computing with lattice surgery," *Quantum*, vol. 3, 2019.

[24] B. M. Terhal, "Quantum error correction for quantum memories," *Reviews of Modern Physics*, 2015.

[25] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, 2021.

[26] Z. Chen, K. J. Satzinger, J. Atalaya, A. N. Korotkov, A. Dunsworth, D. Sank, C. Quintana, M. McEwen, R. Barends, P. V. Klimov, S. Hong, C. Jones, A. Petukhov, D. Kafri, S. Demura, B. Burkett, C. Gidney, A. G. Fowler, H. Putterman, I. Aleiner, F. Arute, K. Arya, R. Babbush, J. C. Bardin, A. Bengtsson, A. Bourassa, M. Broughton, B. B. Buckley, D. A. Buell, N. Bushnell, B. Chiaro, R. Collins, W. Courtney, A. R. Derk, D. Eppens, C. Erickson, E. Farhi, B. Foxen, M. Giustina, J. A. Gross, M. P. Harrigan, S. D. Harrington, J. Hilton, A. Ho, T. Huang, W. J. Huggins, L. B. Ioffe, S. V. Isakov, E. Jeffrey, Z. Jiang, K. Kechedzhi, S. Kim, F. Kostritsa, D. Landhuis, P. Laptev, E. Lucero, O. Martin, J. R. McClean, T. McCourt, X. Mi, K. C. Miao, M. Mohseni, W. Mruczkiewicz, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Newman, M. Y. Niu, T. E. O'Brien, A. Opremcak, E. Ostby, B. Pató, N. Redd, P. Roushan, N. C. Rubin, V. Shvarts, D. Strain, M. Szalay, M. D. Trevithick, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, S. Boixo, V. Smelyanskiy, Y. Chen, A. Megrant, and J. Kelly, "Exponential suppression of bit or phase errors with cyclic error correction," *Nature*, 2021.

[27] A. B. Ziad, A. Zalawadiya, C. Topal, J. Camps, G. P. Gehér, M. P. Stafford, and M. L. Turner, "Local clustering decoder: a fast and adaptive hardware decoder for the surface code," *arXiv preprint arXiv:2411.10343*, 2024.

[28] N. Ofek, A. Petrenko, R. Heeres, P. Reinhold, Z. Leghtas, B. Vlastakis, Y. Liu, L. Frunzio, S. M. Girvin, L. Jiang, and et al., "Extending the lifetime of a quantum bit with error correction in superconducting circuits," *Nature*, 2016.

[29] Y. Xu, G. Huang, N. Fruitwala, A. Rajagopala, R. K. Naik, K. Nowrouzi, D. I. Santiago, and I. Siddiqi, "Qubic 2.0: An extensible open-source qubit control system capable of mid-circuit measurement and feed-forward," *arXiv preprint arXiv:2309.10333*, 2023.

[30] C. A. Ryan, B. R. Johnson, D. Ristè, B. Donovan, and T. A. Ohki, "Hardware for dynamic quantum computing," *Review of Scientific Instruments*, 2017.

[31] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, and et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, 2019.

[32] Quantum Machines. (2023) Accessed: 2023-09-17. [Online]. Available: https://www.quantum-machines.co/resources/brochure/

[33] Zurich Instruments, "Programmable quantum system controller," 2025. [Online]. Available: https://docs.zhinst.com/pdf/ziPQSC_UserManual.pdf

[34] Qblox, "Cluster series mainframe," 2023. [Online]. Available: https://assets-global.website-files.com/653289e64ff83c71222f6bf2/65425680a033f49deaac899b_QBLOX_PRODUCTSHEET_CLUSTER_MAINFRAME_V1_6_1.pdf

[35] Y. Wu, N. Liyanage, and L. Zhong, "LEGO: QEC decoding system architecture for dynamic circuits," *arXiv preprint arXiv:2410.03073*, 2024.

[36] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie, and E. T. Campbell, "Parallel window decoding enables scalable fault tolerant quantum computation," *Nature Communications*, 2023.

[37] X. Tan, F. Zhang, R. Chao, Y. Shi, and J. Chen, "Scalable surface code decoders with parallelization in time," *PRX Quantum*, 2022.

[38] N. Liyanage, Y. Wu, A. Deters, and L. Zhong, "Scalable quantum error correction for surface codes using FPGA," in *Proc. IEEE Int. Conf. Quantum Computing & Engineering (QCE)*, 2023.

[39] *Aurora 64B/66B v12.0 LogiCORE IP Product Guide*, AMD Inc, 11 2023. [Online]. Available: https://docs.amd.com/r/en-US/pg074-aurora-64b66b

[40] "Eos Core," https://github.com/yale-paragon/EosCore, 2024.

[41] "DecoNet : Network-Integrated Decoding System," https://github.com/yale-paragon/DecoNet, 2025.

[42] B. Barber, K. M. Barnes, T. Bialas, O. Buğdaycı, E. T. Campbell, N. I. Gillespie, K. Johar, R. Rajan, A. W. Richardson, L. Skoric, C. Topal, M. L. Turner, and A. B. Ziad, "A real-time, scalable, fast and highly resource efficient decoder for a quantum computer," *arXiv preprint arXiv:2309.05558*, 2023.

[43] "QEC Playground," https://github.com/yuewuo/QEC-Playground, 2023.

[44] A. Holmes, M. R. Jokar, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong, "NISQ+: Boosting quantum computing power by approximating quantum error correction," in *Proc. ACM/IEEE Int. Symp. Computer*

Architecture (ISCA), 2020.

[45] A. G. Fowler and C. Gidney, "Low overhead quantum computation using lattice surgery," *arXiv preprint arXiv:1808.06709*, 2019.