High Throughput Massive MIMO Signal Decoding Using Multi-Level Tree Search on FPGAs

Mohamed W. Hassan, Hatem Ltaief, Suhaib A. Fahmy
King Abdullah University of Science and Technology (KAUST)
Extreme Computing Research Center (ECRC)
Thuwal, Saudi Arabia
{mohamed.hassan,hatem.ltaief,suhaib.fahmy}@kaust.edu.sa

***Abstract*—Supporting the evolution of wireless communication beyond 5G using high-performance networks requires massive device connectivity. Massive Multiple-Input Multiple-Output (MIMO) systems have been used and proven to increase the data throughput of wireless links. However, scaling such systems to a large number of antennas and a high modulation factor entails a significant computational cost for signal decoding using conventional non-linear decoders. Heuristic tree-search-based approaches have been proposed to address this challenge as a means to achieve real-time decoding requirements for large MIMO configurations. FPGAs represent an ideal platform for accelerating MIMO signal decoding on account of their low latency and potential for integration within the signal processing chain while consuming low power. This paper presents software/hardware co-design for a multi-level tree search approach that integrates the computation of multiple tree levels. The proposed heuristic transforms the tree search process into a streaming operation well suited for the FPGA's architecture. We show a series of hardware design and algorithmic optimizations that significantly improve scalability and decoding time, resulting in a design capable of decoding 64×64 64-QAM MIMO within 10ms real-time requirements.**

***Index Terms*—Wireless communication, MIMO, field programmable gate arrays, signal decoding, sphere decoder.**

I. Introduction

Next-generation wireless systems are expected to cater to users' ever-increasing need for high-performance communication. Multiple-Input Multiple-Output (MIMO) systems are a proven solution to maximize wireless capacity [1–7]. While MIMO is already a core component of 4G and 5G communications systems [7], scaling to larger systems within real-time constraints presents a challenge for future adoption.

MIMO systems can scale across two dimensions: the number of antennas and the modulation factor. The critical challenge is the complexity of decoding the signal at the receiver side. Non-linear decoders, such as Maximal Likelihood [8] and Sphere Decoders (SDs) [9, 10], exhibit better Bit Error Rate (BER) performance than linear decoders [6, 11–14], especially when decoding multiple parallel communication streams. The decoding process is mapped to a tree search problem, which grows exponentially, making it computationally complex and hindering scalability. Hence, practical solutions rely on sub-optimal heuristics to reach an approximate solution while reducing complexity [15–19].

Tree-search-based SD is a well-established decoding algorithm that relies on mapping the decoding process to GEMM-based computation [12–14]. Two main challenges prohibit scaling to larger MIMO configurations: real-time decoding constraints and hardware resources constraints. Real-time decoding restricts the decoder's scalability due to the exponential computational complexity of the problem. Hardware resources limit scaling the modulation factor due to the exponential growth of the decoder's memory footprint.

This work tackles these scalability challenges using hardware and algorithmic optimizations. SD includes an inherent computationally expensive backtracking step that bottlenecks the decoding process [14]. This paper proposes an approach that eliminates backtracking from the SD algorithm, making the tree search a stream-forward process well-suited to exploit the FPGA's infrastructure. The proposed heuristic baseline design is thoroughly analyzed, and performance & memory models are used to identify design scalability challenges. Hardware optimizations, in the form of kernel fusion and maximizing data reuse, significantly improve resource utilization and on-chip memory usage. Moreover, the decoding process was accelerated enhancing the decoding throughput, thereby enabling the accommodation of larger MIMO configurations. Additionally, we employ algorithmic refactoring to extract a static portion of the computation into a pre-processing step, significantly reducing the decoding time and improving the decoder's scalability further.

Hence, this paper explores a software/hardware co-design approach to scale hardware-based MIMO decoding systems while limiting the impact of the exponentially increasing computation. The decoder achieves real-time decoding ($10ms$ decoding time and 10^{-2} BER [2, 3, 12]) for massive MIMOs up to the size of 64×64 with 64-QAM modulation. We show significant improvement in decoding time compared to both (GPU) vector- and (FPGA/ASIC) spatial-based designs, as we show in Section V. Moreover, our approach decodes the signal at lower SNR (db) values, requiring less transmission power. While some vector-based decoders exhibit better BER performance, the proposed design significantly outperforms them in terms of decoding time.

The contributions of this work can be summarized as:

- A heuristic to determine the most promising path in a tree search while eliminating the backtracking step, transforming it into a stream-forward operation (Section III).

- A novel software/hardware co-design that maximizes data reuse, which enables the decoding of large-scale MIMO configurations (Section IV).
- A detailed performance/memory model for reasoning about the workload and guiding optimizations. We utilize the models to identify scalability bottlenecks and propose solutions to mitigate them (Sections III and IV).
- The proposed design achieves significant speedup compared to state-of-the-art decoders on various platforms (GPU/FPGA/ASIC). We show orders of magnitude speedup compared to vector-based implementations at the cost of increasing the required transmit power to decode the signal. On the other hand, we show comparable BER performance with spatial-based designs while achieving up to 15× speedup in the decoding time (Section V).

II. Background & Literature Review

A. System Model

A typical MIMO system with M transmitters and N receivers communicating via a channel is shown in Figure 1. The transmitter sends M data streams represented as vector $s = [s_0, s_1, ..., s_{M-1}]$, where s_i belongs to a finite alphabet set of complex constellations denoted by Ω. We consider a small-scale fading channel represented as channel matrix H, which is an N×M matrix where h_{ij} is a complex random variable with mean 0 and variance 1, modeling the fading gain between transmitter j and receiver i. The received signal $y = [y_0, y_1, ..., y_{N-1}]^T$ can be modeled as in Equation 1, where $n = [n_0, n_1, ..., n_{N-1}]^T$ represents additive white Gaussian noise, where n_i is an independent zero-mean circularly symmetric complex Gaussian random variable with variance σ^2.

$$y = Hs + n. \tag{1}$$

B. Sphere Decoding (SD)

Non-linear decoders excel over their linear counterparts in terms of decoding accuracy, especially when scaling the modulation factor and the number of antennas. Non-linear decoding represents the problem as a tree search operation calculating a posterior probability for all possible transmitted vectors $s \in S$, where $|S| = |\Omega|^M$. The resulting vector, s, minimizes the distance between the received vector y and the assumed noiseless vector Hs as shown in Equation 2.

$$\hat{s} = arg\min_{s \in S} ||y - Hs||^2. \tag{2}$$

The SD algorithm enumerates the points inside a hypersphere of radius r around the received point y. The sphere radius r is set initially by the user to prune the search space, as shown in Equation 3; however, it can be subsequently updated at run-time to prune the search space further. The optimization problem shown in Equation 3 is translated into a least-square problem by performing a QR decomposition of the channel matrix $H = QR$, where $Q \in C^{N \times N}$ is an orthogonal matrix

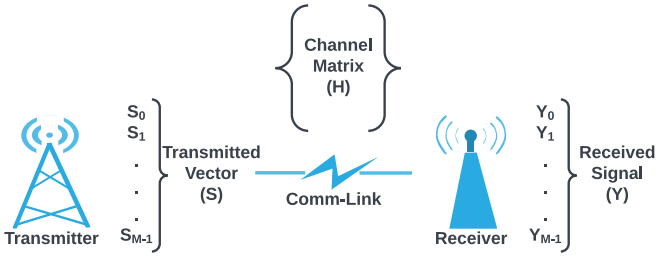

Fig. 1: Typical M×N MIMO system.

and $R \in C^{N \times M}$ is an upper triangular matrix. This translation transforms Equation 3 into Equation 4:

$$||y - Hs||^2 \leq r^2 \tag{3}$$

$$\begin{aligned} ||y - Hs||^2 &= ||y - QRs||^2 \\ &= ||Q(Q^H y - Rs)||^2 \\ &= ||Q^H y - Rs||^2 \\ &= ||\bar{y} - Rs||^2. \end{aligned} \tag{4}$$

C. Vector-Based MIMO Decoders

GPU acceleration of MIMO decoding relies on exploiting the parallelism afforded by GPU cores suited to matrix multiplication. The authors in [12] proposed a GEMM-based variant of the SD algorithm using Breadth-First Search (BFS) traversal to enable dependence-free parallelism that fits the GPU's architecture. However, this approach ultimately increases computational complexity since BFS reaches leaf nodes much later than other tree traversal strategies.

The design in [5] is a Fixed Complexity Sphere Decoder (FCSD) that sets a threshold for the number of independent paths traversed in the search tree. While other FCSD implementations exploit the GPU's massive parallelism by instantiating multiple decoders [20], the design in [5] parallelizes the decoding algorithm itself. The CPU host orchestrates and schedules operations while exploiting the cuBLAS library to accelerate the compute-intensive operations on the GPU. The design was run on an Nvidia GTX 760 GPU and achieved significant performance improvement over CPU implementations without jeopardizing BER performance. The decoder proposed in [5] scaled to 16×16 MIMO with 64-QAM modulation factor.

A flexible N-Way MIMO detector proposed in [15] decomposes the decoding process into QR decomposition followed by a tree search. The authors utilized the GPU's parallelism to perform multiple tree searches in parallel, greatly enhancing BER performance. Unlike conventional decomposition methods, the QR decomposition kernel in [15] is optimized to process many small dense matrices in parallel. Increasing parallelism improved BER performance with no impact on decoding time. The design was run on an Nvidia GeForce GTX 690 and scaled to 4×4 MIMO with 64-QAM.

We observed that vector-based solutions use parallelism to overlap multiple distinct instances of the tree search. While such an approach yields good BER performance [5, 15], it does not improve the latency of the decoding process, which depends on algorithmic properties and how well they map to the underlying architecture, especially when scaling the number of antennas. As such, spatial-based decoders have the potential to scale well without compromising BER performance through enhanced specialization.

D. Spatial-Based MIMO Decoders

Low latency and low power decoders are desirable for meeting the latency requirements of signal decoding within the power envelope of wireless base stations. Hence, the implementation of MIMO decoders on spatial architectures (FPGAs/ASICs) has been explored. The research proposed in [21] divides the decoding process and associated data structures on a CPU-FPGA platform. Optimizing the data flow between parallel threads on the CPU and pipelined kernels on the FPGA allows the design to scale and improve performance compared to CPU-based MIMO decoders. The configurable design in [22] employs a Robust Bounded Spanning with Fast Enumeration (R-BSFE) approach to maintain accuracy and reduce complexity. Moreover, a channel matrix reordering technique is used to enhance the accuracy, reducing the required transmission power by 5 dB. This design was tested up to 8×8 MIMO and 1024-QAM. However, the BER performance falls under the required 10^{-2} threshold at high SNR values, as is shown later in Section V. A GEMM-based SD implementation was proposed in [14] and implemented on a Xilinx U280 FPGA. The authors combined the tree traversal approach proposed by [23] with BLAS-based tree construction [12]. A dynamically built search tree avoids communication with the host, allowing performance to scale to 10×10 MIMO with 16-QAM modulation.

ASICs offer the advantage of customized hardware architecture, which can run at higher frequencies than FPGAs. In [24], a modified version of Dijkstra's algorithm is used to implement a MIMO decoder on a $0.49mm^2$, 25.1K-gate ASIC. Dijkstra's algorithm is modified to enable overlapped computation with the best candidate node pre-calculated to be expanded in the next iteration, effectively reducing the critical path delay. The ASIC presented in [24] is a 4×4 MIMO 16-QAM decoder that decodes the signal at a throughput of 302 Mb/s at an SNR of 20 dB. Other work in [25] implements a Monte Carlo tree search (MCTS)-based MIMO decoder on an ASIC, then uses the same approach for antenna selection in [26]. Their approach eliminates costly matrix multiply operations and applies optimizations to limit the search space without sacrificing BER performance. They developed a 1.43 mm^2 65nm CMOS MIMO decoder for 64×8 MIMO with 16-QAM delivering up to 665Mb/s of throughput.

A key observation from the literature review is that spatial-based decoders scale better than vector-based decoders (supporting a higher number of antennas and modulation factors). While spatial-based decoders show significant improvement in decoding time (i.e., throughput) compared to GPU decoders, the transmit power required to decode the signal with 10^{-2} BER is higher (i.e., higher SNR). This previous work demonstrates that scalability remains a challenge and that beyond parallelism, algorithmic modifications are required to achieve the required decoding time and BER for complex MIMO systems. Our detailed algorithmic model in Section III helps guide us to such an architecture.

III. Multi-Level Tree Search for Signal Decoding

Here, we explain the multi-level heuristic proposed in this paper to traverse the tree search space efficiently. The algorithm is explained, and its complexity analysis and memory-usage models are presented. This is particularly important for evaluating the scalability boundaries of the hardware design.

A. Multi-Level Heuristic

The proposed heuristic constructs the tree search space and models the MIMO system similar to the SD algorithm explained in Section II-B. However, the critical differences are in the way the search tree is ***traveresed***, and the way ***unpromising branches are pruned***. The multi-level heuristic traverses the tree search space L levels at a time. One global timestep explores all the nodes generated from one root node within L levels of the tree. The leaf node with the lowest PD evaluation in one global timestep is chosen as the next root node. The remaining leaf nodes are discarded, as shown in Figure 2. Higher L values entail exploring more nodes at each global timestep, which in turn covers a higher portion of the search space, yielding better BER performance. However, this comes at the cost of a higher computational workload that impacts throughput.

Each global timestep includes four main functions as shown in Algorithm 1: (1) *Load* loads the corresponding input matrices Y_{step}, H_{step}, (2) *Expand* enumerates all successors at level L, (3) *GEMM* evaluates the PD for all generated nodes, (4) *Norm* normalizes the results and chooses the node with minimum PD to be the next root.

Algorithm 1: Multi-Level Tree Search Algorithm

Data:
Received signal (Y)
Channel matrix (H)
Combined levels (L)
Result:
Decoded signal vector ($\hat{s}$)

```
Node ← root;
for i : 0 → Steps do
    Y_i, H_i ←Load(Y, H) ;    /* Load inputs */
    B ←Expand(L) ;  /* Generate subtree */
    G ←GEMM(H, B, Y);
    Node ←Norm(G) ;  /* Norm & min(PD) */
end
```

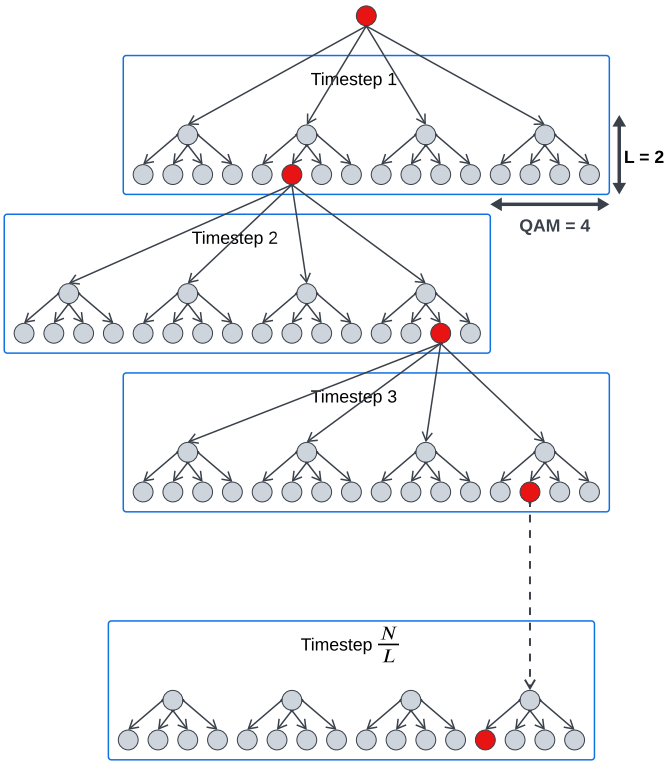

Fig. 2: Tree expansion and selection process based on the proposed heuristic for a 4-QAM MIMO configuration, where $L = 2$, combining two levels to process per global timestep. Red nodes have the lowest PD in their respective timesteps.

1) Load: The channel matrix (H) and received signal (Y) act as inputs to the computing kernel. The received signal (Y) is a duplicated vector that constructs the matrix for GEMM computation. Each level of the search tree is represented by a part of these matrices. Hence, only a slice of these matrices is loaded for each global timestep. Start and end indices are calculated to determine the corresponding chunk of data to fetch depending on the node's level.

2) Expand: This function generates all the successor nodes to the current tree root. All children inherit the decoded portion of the signal that the current root holds, which initially starts as an empty set. Using the current state of the tree, the *Expand* function constructs the matrix (B) to model the recovered signal so far and enumerate all possible values for the next symbol to decode. The algorithmic complexity of this function is $\mathcal{O}(L \times SubTree_L)$, where $SubTree_L = q^L$ is the size of the subtree. The size of the matrix (B) and its implications will be discussed in the memory model in Section III-B

3) GEMM: The GEMM engine performs the usual matrix multiply operation, where $Y = (H \times B) + Y$. The output (Y) is stored in matrix (G), which is used to evaluate (PD) values for the leaves of the current subtree. The algorithmic complexity of the GEMM operations is $\mathcal{O}(L \times N \times SubTree_L) \equiv (L \times N \times q^L)$. Increasing the number of levels (L) or the modulation factor (QAM) significantly impacts the complexity of the GEMM operation.

TABLE I: Compute and memory model parameters.

Parameter	Values	Description
N	8,16,32,64	Number of receivers
L	2,4	Number of levels
q	4,16,64	QAM modulation factor
$Steps$	$\frac{N}{L}$	Number of global timesteps
$Tree_A$	$q^{N+1} - 1$	Total number of nodes in the tree
$Tree_L$	q^N	Number of leaf nodes in the tree
$SubTree_A$	$q^{L+1} - 1$	Total number of nodes in the sub-tree
$SubTree_L$	q^L	Number of leaf nodes in the sub-tree
H	Channel matrix	
Y	Received signal	
B	Enumeration of all candidate leaf nodes	
G	GEMM result for all leaf nodes (pre-normalization)	
PD	Partial Distance evaluation for all leaf nodes	

4) Norm: The output of the GEMM operation is normalized to calculate the final (PD) for each leaf node. Nodes are sorted according to their (PD) values to retrieve the lowest (PD), corresponding to the most promising node in this subtree that is more likely to lead to the solution. This function performs floating point multiply and accumulates with a complexity of $\mathcal{O}(L \times SubTree_L)$. The output of this function is the new root node to be processed in the next timestep or the leaf node that represents the decoded signal.

B. Memory Model

We established in Section III-A that the proposed algorithm is more predictable than other non-linear decoders, so its performance can be statically estimated using the model explained here. Table I defines some parameters for the model that are used throughout the paper. Scalability is the primary design objective in this work, hence the focus on the memory footprint of the algorithm. Implementing a hardware accelerator for such an exponentially complex problem incurs presents a challenge due to the limited resources available. Five main data structures are used to implement the proposed algorithm in hardware, two of which are inputs offloaded to the accelerator and three holding intermediate data. We should also consider the fact that in a real deployment, CPUs and GPUs would require the data to be moved into memory and then accessed by processing cores. FPGAs, however, are equipped with high-speed transceivers capable of delivering signal data directly into the computational pipeline.

1) Search space: The tree depth depends on the number of receivers (N), and the tree width depends on the modulation factor (QAM), so the total number of nodes in the tree can be evaluated as ($q^{N+1} - 1$). Pruning the search space is essential to achieve reasonable decoding times. The SD algorithm is data-dependent, with pruning based on each node's PD compared to the sphere radius. Such a pruning strategy cannot be statically quantified; however, our proposed heuristic has a statically defined pruning strategy. Empirical evaluation shows that the SD algorithm explores ($< 1\%$) of the search space. Our proposed approach explores ($SubTree_A \times Steps$), where $SubTree_A$ is the total number of nodes in (L) levels and ($Steps$) is the number of global timesteps required to reach the

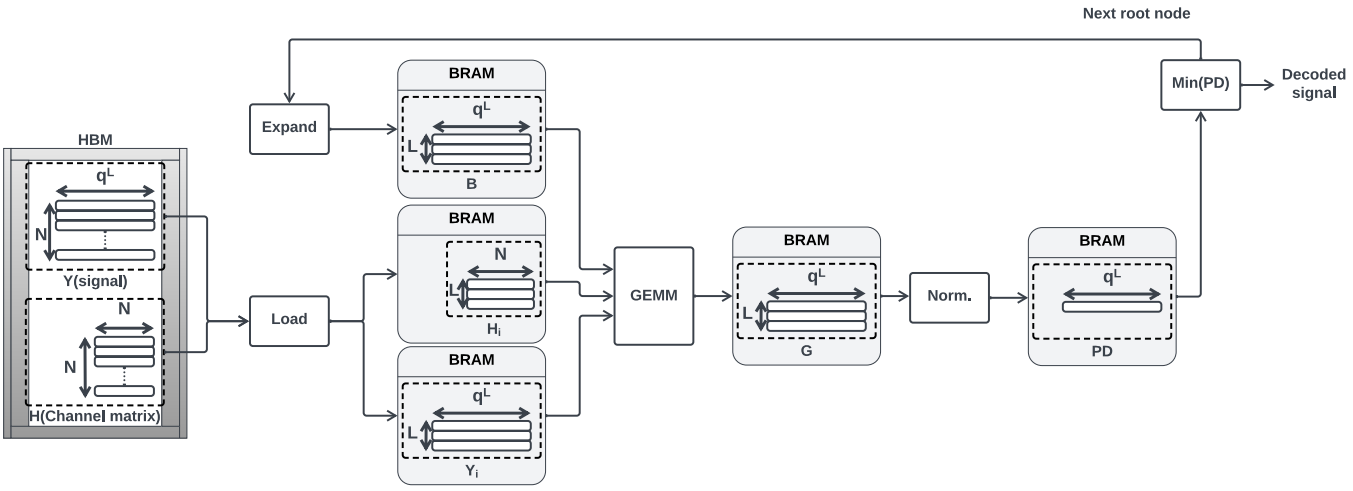

Fig. 3: Dataflow hardware architecture for the baseline (V1) decoder.

leaf nodes. Hence, the proposed algorithm explores ($\ll 1\%$) of the search space.

2) Input data structures: Matrices ($H\&Y$) are two inputs that are moved from the host to the accelerator, and their memory complexity is shown in Equations 5 & 6. The matrix (H) models the communication channel and, hence, does not change between runs. On the other hand, the vector (Y) represents the incoming signal to be decoded, which is streamed in from deployed antennas. In the case of CPU & GPU decoding, the received signal has to be passed to the main memory and then fetched by the accelerator. In contrast, RF-soc-enabled FPGA can employ high-speed transceivers to feed data directly into the decoding pipeline, bypassing the memory access step.

$$mem(H) = \mathcal{O}(N^2) \tag{5}$$

$$\begin{aligned} mem(Y) &= \mathcal{O}(N \times SubTree_L) \\ &= \mathcal{O}(N \times q^L) \end{aligned} \tag{6}$$

The algorithm operates only on a chunk of these matrices for each global timestep. Equations 7 & 8 show the resulting memory complexity for each global timestep.

$$mem(H_{step}) = \mathcal{O}(N \times L) \tag{7}$$

$$mem(Y_{step}) = \mathcal{O}(L \times q^L) \tag{8}$$

3) Intermediate Data Structures: The size of the intermediate data structure detailed earlier in Section III-A is critical. Limited on-chip memory restricts the scalability of the design, so the size of these data structures must be optimized. The three major data structures are (B, G, PD), with memory complexity shown in Equations 9, 10, and 11.

$$mem(B) = \mathcal{O}(N \times SubTree_L) \tag{9}$$

$$mem(G) = \mathcal{O}(L \times SubTree_L) \tag{10}$$

$$mem(PD) = \mathcal{O}(SubTree_L) \tag{11}$$

It is worth noting that the size of the subtree ($SubTree_L$) has a significant impact on the memory footprint. For 64×64 MIMO with $q = 64$, the subtree size is $SubTree_L = 64^{64}$. A data structure of that size cannot fit in the limited on-chip memory of the FPGA.

IV. Multi-Level Tree Search Hardware Design

Increasing the throughput of wireless networks using MIMO largely depends on scaling the number of antennas and the modulation factor. However, due to the computationally challenging nature of the signal decoding process, it would be infeasible to target a fully scalable system. We target a challenging configuration of 64×64 MIMO using 64-QAM modulation while adhering to a BER of 10^{-2}. Two main constraints restrict our system's design space: (a) hardware resources limitation, i.e., decoder design fits in the FPGA, and (b) real-time decoding constraint, i.e., decoding the signal in $\leq 10ms$. The design stages detailed in this section address these constraints in order to achieve the objective of supporting real-time decoding for a 64-QAM 64×64 MIMO system.

A. Baseline (V1)

1) Design: The baseline design maximizes the use of on-chip BRAM to exploit its single-cycle access latency. Initially, the channel matrix (H) is stored in HBM, while the received signal (Y) is streamed from the RF baseband antennas. In our experiments, we emulate the MIMO operation using Monte Carlo simulation, hence we store the simulated received signal (Y) in HBM. Accesses are ordered as would be the case

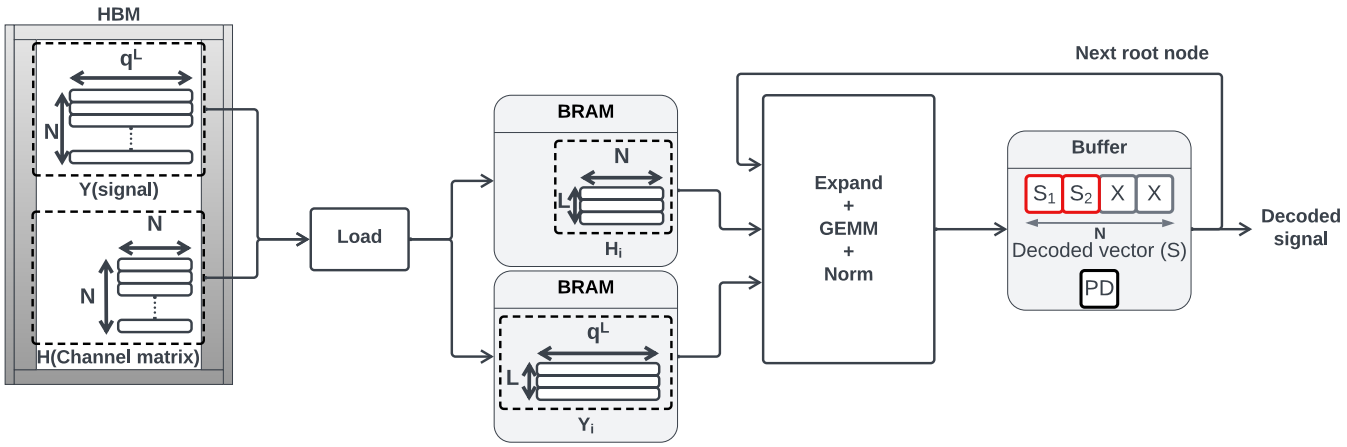

Fig. 4: Hardware architecture for the hardware optimized (V2) decoder.

in a live RF stream. The (Load) function fetches the global timestep's corresponding chunk of the channel matrix (H) and the received signal (Y) from HBM and buffers them in on-chip BRAMs. The (Expand) function enumerates all successor nodes in (L) levels from the root based on the modulation factor (QAM) and stores the resulting matrix (B) in BRAM. Consequently, the GEMM engine accesses all its operands from local on-chip memory, as illustrated in Figure 3. Finally, the normalization step (Norm) evaluates the (PD) for all active nodes and chooses the minimum PD to be the new root. The memory complexity of each module is shown in Figure 3.

This architecture is well suited for dataflow optimization; however, its high BRAM usage prohibits scaling. The decoding times for 4-QAM & 16-QAM configurations are shown in Figure 8. The baseline design achieves real-time decoding for all 4-QAM configurations; however, it breaks the real-time decoding constraint for 16-QAM configurations of sizes larger than 16×16. Moreover, the design cannot scale to 64-QAM due to resource utilization constraints. Increasing the modulation factor (QAM) has an exponential impact on the BRAM usage to store matrix (B).

2) Algorithmic complexity: Due to the stream-forward nature of the proposed heuristic, the computational complexity can be statically estimated. The overall complexity of the hardware kernel (T_{V1}) can be estimated using Equation 12, where T_{prep} represents the (Load) and (Expand) functions as they are executed in parallel. The complexity breakdown is shown in Equations 13, 14, and 15.

$$T_{V1} = \sum_{i=1}^{Steps} T_{prep} + T_{gemm} + T_{norm} \tag{12}$$

$$T_{prep} = \mathcal{O}(L \times q^L) \tag{13}$$

$$T_{gemm} = \mathcal{O}(L \times N \times q^L) \tag{14}$$

$$T_{norm} = \mathcal{O}(L \times q^L) \tag{15}$$

B. Hardware kernel fusion (V2)

1) Design: It can be noticed that data structures with a memory footprint of ($L \times q^L$) present a bottleneck for on-chip memory usage. Hence, in this design, we optimize data reuse, saving memory usage as well as optimizing performance, as shown in Figure 4.

Fusing the *Expand* and *GEMM* kernels together eliminates the need for a data transfer of size ($L \times q^L$). Moreover, we eliminate the need to store the (B) enumeration matrix. We exploit the fine granularity of the reconfigurable logic to customize computation to generate a chunk of columns of the (B) matrix, then perform all the necessary multiply-accumulate operations with the channel matrix (H) storing a partial evaluation in the resulting (G) matrix.

We further fuse the *Norm* process to eliminate the need for storing the resulting G, maximizing data reuse. We eliminate the calculation of all the G matrix and the associated normalization required to obtain the PD evaluation of all the nodes in a subtree. Instead, we transform the operation into a vector product followed by normalization directly. Hence, each iteration generates the PD for a node directly. We only store one node (minimum PD) instead of $L \times q^L$ nodes. This approach also eliminates iterating over the active nodes to find the minimum PD, reducing algorithmic complexity further.

This transformation fuses the *GEMM* and *Norm* operations inside the expansion process, allowing fine-grained optimizations in the hardware design. The fusion optimizes memory usage as well as enhances decoding time. The V2 design eliminates the storage of G and PD matrices, significantly enhancing scalability. It also generates the B-matrix in chunks that are processed on the fly, reducing on-chip memory usage. Figure 8 shows the decoding time compared to the baseline (V1), where V2 enables 16-QAM configurations to be decoded in real-time. While V2 supports 64-QAM configuration, its computational complexity makes real-time decoding impossible. Hence, algorithmic refactoring is required to allow 64-QAM configurations to be decoded in real-time.

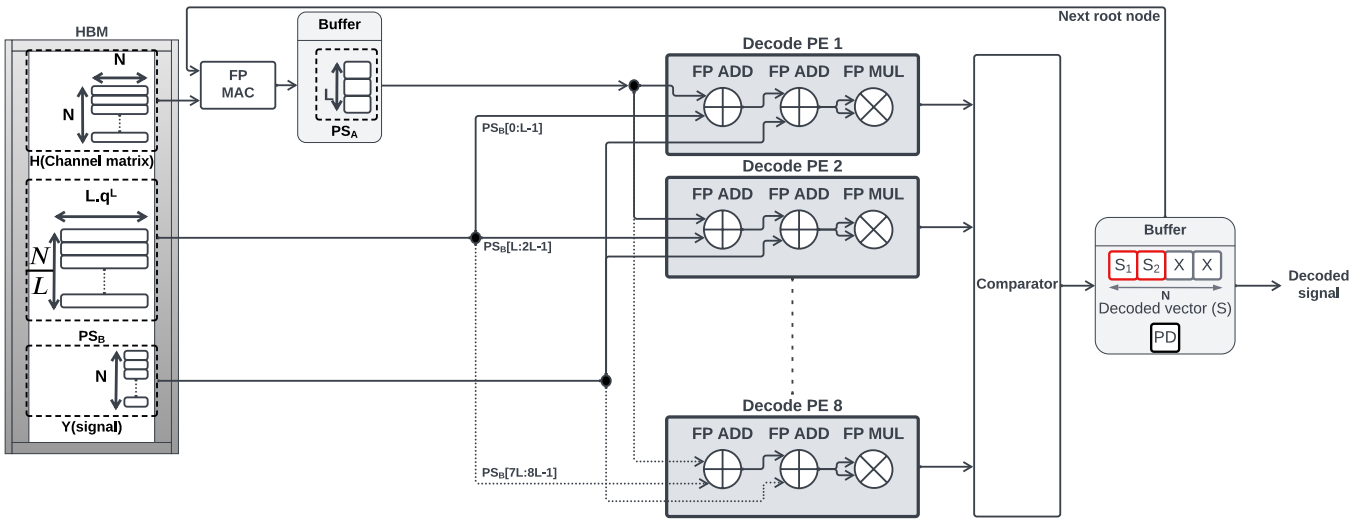

Fig. 5: Hardware architecture of algorithmic refactored (V3) decoder.

2) Algorithmic complexity: This design iteration tackles on-chip memory usage by fusing kernels to eliminate the need for on-chip buffering of multiple data structures, as well as eliminating the need to stream the data from one kernel to the other. While these optimizations significantly improve both performance and memory usage, the algorithm's theoretical complexity remains unchanged from the baseline.

C. Algorithmic refactoring (V3)

1) Design: Scaling the modulation to 64-QAM incurs a memory explosion that requires further algorithmic refactoring. Each global timestep computes the partial distance evaluation (PD) of all leaf nodes in the subtree of (L) levels. This process involves performing the GEMM operation for all $SubTree_L$ leaf nodes. Two observations were used to drive the algorithmic refactoring: the channel matrix (H) is fixed for a given channel, and the enumeration matrix (B) is fixed for the modulation scheme. The channel matrix (H) rarely changes, for instance, due to the movement of the radio system or server weather conditions. Hence, this process can be extracted out of the computational pipeline and moved to pre-processing.

Figure 6 explains the portion moved to pre-processing. This data is readily available before processing starts and is static across iterations. A chunk of the channel matrix is multiplied by the enumerated B-matrix at each global timestep. The partial sum of the GEMM operation is buffered in the FPGA's HBM to be streamed into the computational pipeline. This algorithmic refactoring eliminates the complexity of the GEMM operation, which can be estimated using Equation 14. Hence, the total number of MAC operations in the computational pipeline is significantly reduced. Moreover, eliminating the GEMM operation eliminates the need to duplicate the received signal (Y), reducing its memory footprint.

After pre-computing the partial sum, PS_B, the remaining portion of the GEMM operation is shown in Figure 7. The remaining computation uses the symbols that were previously decoded from previous global timesteps. These recovered symbols are augmented with the enumerated B-matrix for use in the GEMM with channel matrix (H). This portion can be done once at the start of the global timestep, as shown in Figure 5, instead of repeating it for all leaf nodes. Hence, another partial sum is computed at the start of the global timestep and buffered in on-chip BRAM. The buffered partial sum is reused for all leaf nodes until the global timestep

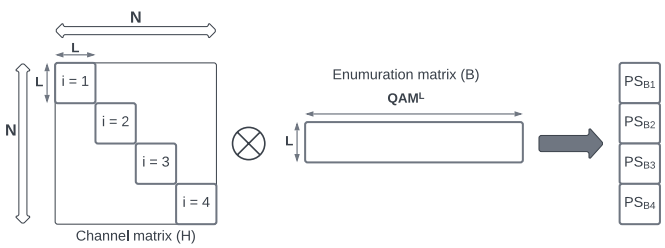

Fig. 6: Preprocessing computation. A block of size $L \times L$ from the channel matrix (H) is multiplied by the enumeration (B) matrix to generate partial sum results for each global timestep.

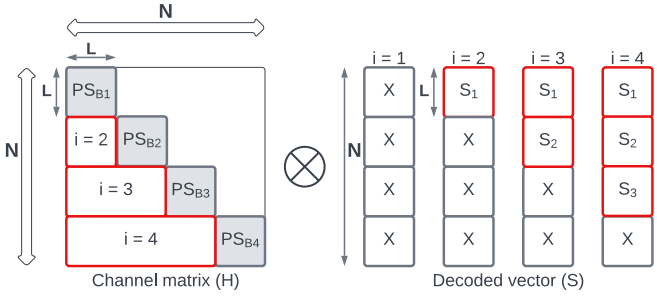

Fig. 7: Hardware kernel computation for four timesteps. Partial sum results from preprocessing (PS_B) are used along with the decoded portion of the signal to compute the (PD) value.

Algorithm 2: Multi-Level Tree Search Algorithm (V3)

Data:
Received signal (Y)
Channel matrix (H)
Combined levels (L)
Enumeration partial sum (PS_B)
Result:
Decoded signal vector ($\hat{s}$)

```
Cur_Node ← root;
for i : 0 → Steps do
    Y_i, H_i, PS_B(i) ←Load(Y, H, PS_B(i));
    PS_A(i) ←Matvec(H_i,ŝ);
    Cur_Node ←Decode(PS_A(i), PS_B(i), Y_i);
end
```

concludes.

Details of the hardware kernel are shown in Figure 5, and the refactored algorithm is shown in Algorithm 2. The sub-tree root node is stored in BRAM with a dedicated memory space (S vector) to hold the decoded symbols of the received signal. Recovered symbols and the corresponding chunk of the channel matrix (H) are input to a floating point MAC unit. The resulting partial sum (PS_A) is buffered in BRAM to be reused. The *Decode PE* shown in Figure 5 is responsible for evaluating the PD of leaf nodes by performing two floating-point adds and one floating-point multiply. Multiple decoding PEs are instantiated to evaluate multiple leaf nodes in parallel. The results of the *Decode PEs* pass through a comparator that determines the minimum (PD) node to be used as the root node for the next global timestep.

2) Algorithmic complexity: The GEMM computation has been broken down into the summation of two partial sums (one in pre-processing and one in the hardware kernel). The hardware kernel iterates over the $SubTree_L$ leaf nodes to perform only two floating point additions and one MAC operation instead of a full GEMM. The overall complexity of the hardware kernel (T_{V3}) can be estimated using Equation 16, with the breakdown in Equations 17, and 18.

$$T_{V3} = \sum_{i=1}^{Steps} T_{Matvec} + T_{Decode} \tag{16}$$

$$T_{Matvec} = \mathcal{O}(L \times N) \tag{17}$$

$$T_{Decode} = \mathcal{O}(L \times q^L) \tag{18}$$

D. Performance Discussion

Table II summarizes the achievable support of each design stage for all tested MIMO configurations. The baseline design (V1) supports 4-QAM configurations for all antenna sizes up to 64×64. However, the modulation factor significantly impacts the exponentiality of the tree search's computational complexity. Consequently, when decoding 16-QAM configurations, (V1) breaks the real-time constraint for MIMO sizes larger than 16×16. Moreover, due to the high memory footprint of (V1), 64-QAM configurations are not synthesizable by cause of insufficient resources.

TABLE II: Feasibility of designs for different MIMO configurations. ● indicates design implemented and meeting real time constraints (∗ close to real time constraint). ◐ indicates design does not meet real-time constraints. ○ indicates design does not fit device resources.

Config.	4-QAM			16-QAM			64-QAM		
	V1	V2	V3	V1	V2	V3	V1	V2	V3
8 × 8	●	●	●	●	●	●	○	●	●
16 × 16	●	●	●	●	●	●	○	◐	●
32 × 32	●	●	●	◐	●	●	○	◐	●
64 × 64	●	●	●	◐	●*	●	○	◐	●*

The hardware kernel fusion employed in (V2) optimizes resource utilization, as shown in Table III. These resource savings enable the design to fit the 64-QAM configuration in the FPGA's available resources. Moreover, reducing intra-kernel data communication and maximizing data reuse accelerates the decoding process by an average of 3.2× compared to (V1) for the 16-QAM configuration, as shown in Figure 8. While the (V2) design satisfies the resource constraint in the case of 64-QAM configuration, it fails the real-time decoding constraint for MIMO sizes larger than 8×8, as indicated in Table II.

The algorithmic refactoring in (V3) enables design scalability to support real-time decoding for 64-QAM configurations. The partial evaluation strategy explained in Section IV-C and the parallel decoding PEs deliver accelerated performance. With 42× and 8× speedups compared to (V1) and (V2), respectively, (V3) satisfies all the objectives and constraints defined earlier. This final design stage decodes in real-time MIMO configurations up to 64-QAM 64×64. We successfully

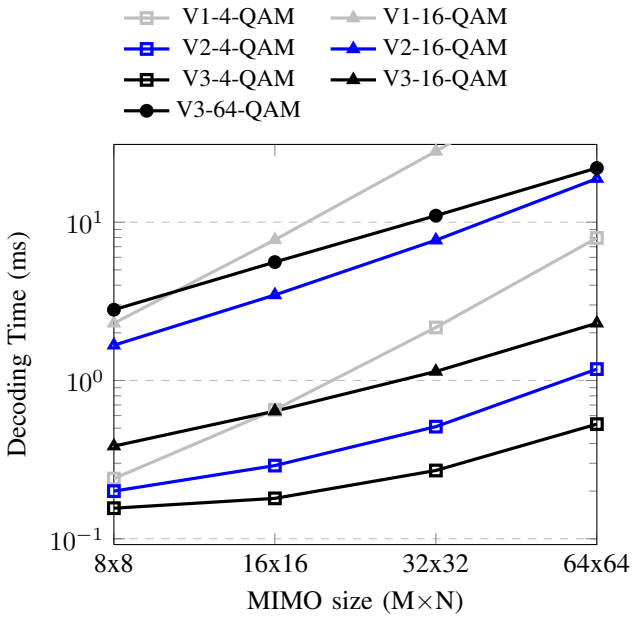

Fig. 8: Decoding time comparison of the design stages, ($L = 2$).

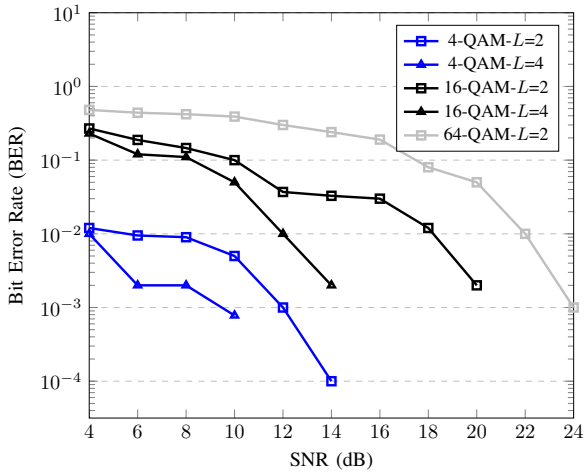

Fig. 9: Bit Error Rate (BER) comparison for V3 implementation. MIMO size is set to 64×64 and tested for $L = 2$ and $L = 4$.

TABLE III: Resource utilization for all design versions synthesized on the Alveo U280 FPGA.

Design	M×N	**LUTs**	**FFs**	**BRAMs**	**DSPs**
V1 (16-QAM)	64×64	35%	24%	29%	2%
V2 (64-QAM)	64×64	28%	19%	18%	2%
V3 (64-QAM)	64×64	20%	17%	18%	2%

scaled the design for more antennas, up to 256×256; however, decoding time exceeded the real-time constraints. On the other hand, scaling the modulation factor to 256-QAM results in a design that exceeds the FPGA's resources due to exponential growth in the on-chip memory requirements.

We further evaluate BER performance and decoding time when increasing the number of levels L from 2 to 4, setting the MIMO size to 64×64. We test the 4-QAM and the 16-QAM configurations of the (V3) design; however, the 64-QAM configuration is tested only with $L = 2$, as shown in Figure 9. Processing four levels with 64-QAM modulation shows infeasible decoding times. We observe a significant increase in computational complexity when increasing the number of levels explored, which outweighs the BER performance improvement. For instance, increasing L for the 16-QAM configuration lowers the SNR at which the signal is decoded from 18 dB to 12 dB, as shown in Figure 9, but decoding time is two orders of magnitude slower.

V. Experimental Evaluation

A. Experemintal Setup

Our experimental platform comprises a Xilinx Alveo U280 FPGA card equipped with 8GB of High Bandwidth Memory (HBM) accessible over 32 channels in addition to 32 GB of DDR4 memory [27]. The card is hosted in a workstation with an AMD Ryzen Threadripper Pro 3975WX 32-core CPU. The FPGA designs are implemented using OpenCL/C++ High-Level Synthesis (HLS) and synthesized using Xilinx Vitis 2020.2, with the designs running at an approximate frequency of 300 MHz. The workstation runs Ubuntu 18.04. We use the Intel Math Kernel Library (MKL) and the Boost library for host-side computation. The test dataset is randomly generated using Monte Carlo simulations to emulate the MIMO system as in [12, 14], to be able to control for a range of different SNR and antenna configurations, as is standard practice in such evaluations. Building a physical testbed to collect real world data for these test scenarios is very challenging considering the complexity and variety of radio hardware required.

B. GPU Comparison

GPU-based designs usually rely on instantiating multiple instances of the tree search to cover more of the tree, enhancing BER performance and increasing utilization of the GPU cores. While this helps enhance BER accuracy, it does not help with latency. Hence, the low number of antennas supported by GPU designs: 32×32 for 16-QAM and 16×16 for 64-QAM as shown in Table V. Tables IV and V show the SNR value at which the BER drops below the threshold of 10^{-2}; a lower value is preferred. The GPU implementations in [5, 15] can decode signals within the acceptable BER threshold at lower transmit power (SNR) than our decoder (and other spatial-based implementations). However, the GPU consumes significantly more power, and the decoding time/throughput of GPU-based designs does not scale to larger configurations.

We compare against the faster Nvidia GTX 690 reported results for the N-way design in [15] as shown in Table IV. Our design is slower for smaller configurations of 4×4 MIMO at 16-QAM since the small size of the search tree does not fully exploit the capabilities of our proposed design. The 64-QAM configuration, however, shows an 8.7× speedup for our design compared to [15]. Moreover, if the GPU design could be scaled to 64×64 MIMO at 64-QAM, our design would have shown even more significant gains.

The authors in [5] attempted to scale the number of antennas, which is reflected in the exponentially slowing decoding times shown in Table V. Our proposed decoder scales the design to a high number of antennas without significant performance degradation. While our design exhibits higher

TABLE IV: Comparison with N-Way-GPU detector [15].

MIMO Config.	N-Way [15] **T'put** (Mb/s)	**SNR** (dB)	Our work **T'put** (Mb/s)	**SNR** (dB)	**Speedup**
4×4 16-QAM	1036.8	10	410	18	0.4×
4×4 64-QAM	519.2	15	4494	22	8.7×

TABLE V: Comparison with FCSD-GPU [5].

MIMO config	**FCSD-GPU** [5] **Time** (ms)	**SNR** (dB)	**Our work** **Time** (ms)	**SNR** (dB)	**Speedup**
8×8 16-QAM	1.32×10^4	14	3.85×10^{-1}	18	10^5
16×16 16-QAM	2.68×10^4	14	6.4×10^{-1}	18	10^5
32×32 16-QAM	4.98×10^6	14	1.14	18	10^6
8×8 64-QAM	1.22×10^4	18	2.8	22	10^4
16×16 64-QAM	4.68×10^5	18	5.6	22	10^5

SNR values at which the signal is decoded, we achieve up to five orders of magnitude speedup compared to [5]. Due to the slow CPU-GPU interconnect, the GPU's design scalability is hindered when the number of antennas grows beyond 4×4. On the other hand, our design scales smoothly up to 64-QAM 64×64 MIMO configurations.

The GEMM-based decoding on GPUs proposed in [12] was reproduced in [14] on Nvidia A100 GPUs and tested for 4-QAM modulations. The decoder traverses the search tree using BFS to maximize the utilization of GPU cores with dependence-free parallelism. However, this method results in more computational complexity as it visits an exponentially larger number of nodes. Our proposed architecture shows an average speedup of 188× decoding 10×10 MIMO at 4-QAM compared to [12] implemented on an Nvidia A100 GPU.

While GPU designs can scale to 64-QAM modulation, the number of antennas does not scale feasibly beyond 4×4. On the other hand, our design scales up to 64×64 MIMO with 64-QAM modulation. This is emphasized by the increasing speedup our design shows compared to different GPU decoders when increasing the number of antennas.

C. FPGA and ASIC Comparison

The FPGA design proposed in [14] uses SD with K-best sorting starting with a relatively large radius, which results in acceptable BER performance at lower SNR as depicted in Figure 10. However, this comes at the cost of higher computing complexity. The maximum configuration supported is 10×10 at 16-QAM. Our proposed design decodes the signal for the same configuration in $0.4ms$, yielding a 40× speedup, which grows further when scaling to 64-QAM, which was not achievable in [14]. Figure 10 shows that our design has comparable BER to other ASIC and FPGA-based designs. However, Table VI shows that we significantly improved the decoding throughput. The speedup column shows the speedup of all designs compared to the slowest, which is the modified

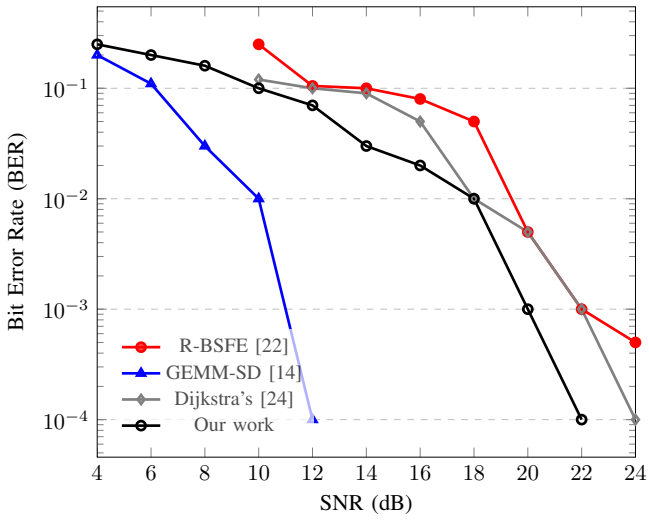

Fig. 10: Bit Error Rate (BER) comparison for multiple spatial-based implementations, 4×4 MIMO 16-QAM.

TABLE VI: Throughput comparison of spatial-based MIMO decoders.

HW	Architecture	Configuration	Throughput (Mb/s)	Speedup
ASIC	Dijkstra's [24]	4×4 16-QAM	302	1×
	MCTS [25]	64×8 16-QAM	665	2.2×
FPGA	R-BSFE [22]	4×4 16-QAM	519	1.7×
		4×4 64-QAM	494	1.6×
	Our work	4×4 16-QAM	410	1.35×
		4×4 64-QAM	4494	14.9×
		64×64 64-QAM	4575	15.1×

Dijkstra ASIC decoder [24]. The R-BSFE ASIC design [22] is 1.3× faster than our proposed design for 4×4 MIMO at 16-QAM due to the small problem size, which fails to exploit the parallelism optimizations in our design. However, scaling our design to 4×4 64-QAM shows around a 9× advantage. The closest reported configuration in other work to our target is the 64×8 64-QAM decoder implemented as MCTS on FPGA in [25]. Even with a larger number of antennas, we achieve a speedup of 6.9× compared to [25].

VI. Conclusion

This work tackles the scaling of Massive MIMO decoding for a large number of antennas and high modulation factor within real-time constraints. We presented an iterative design refinement for a signal decoding architecture that enables scalability and significantly accelerates decoding time. A multi-level tree search heuristic was proposed that eliminates the backtracking step that usually hinders tree search. This effectively transforms the computation required for the tree search into a streaming operation. Hardware optimizations, such as kernel fusion, were applied to maximize data reuse and significantly reduce on-chip memory usage, allowing the design to scale. Moreover, algorithmic refactoring improved decoding throughput, further enabling 64×64 64-QAM MIMO configurations to be supported within real-time constraints.

Our final design achieved significant improvement in both decoding time and scalability, compared to GPU, FPGA, and ASIC decoders at the cost of slightly lower BER accuracy. For the largest 64×64 64-QAM MIMO configuration, we demonstrated up to 15× speedup compared to state-of-the-art spatial-based decoders and 188× speedup compared to the reproduction of a vector-based GPU decoder on modern A100 GPU. BER accuracy can be improved in future work by instantiating multiple tree search instances like GPU approaches, which increases coverage of the search tree while retaining the latency advantage. We are investigating integration with a physical testbed to evaluate the additional latency benefit of direct ingestion of RF data into the FPGA as opposed to a more traditional offload from a host as is required for GPUs. Such integrations are typically done in a power-constrained environment, where our FPGA approach should be much more practical than a GPU.

REFERENCES

[1] T. L. Marzetta, "Massive MIMO: an introduction," *Bell Labs Technical Journal*, vol. 20, pp. 11–22, 2015.

[2] C. Cox, *An introduction to LTE: LTE, LTE-advanced, SAE and 4G mobile communications*. John Wiley & Sons, 2012.

[3] "IEEE standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.

[4] S. Dang, O. Amin, B. Shihada, and M.-S. Alouini, "What Should 6G Be?" *Nature Electronics*, vol. 3, no. 1, pp. 20–29, Jan 2020. [Online]. Available: https://doi.org/10.1038/s41928-019-0355-6

[5] T. Chen and H. Leib, "GPU acceleration for fixed complexity sphere decoder in large MIMO uplink systems," in *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2015.

[6] K. Nikitopoulos, G. Georgis, C. Jayawardena, D. Chatzipanagiotis, and R. Tafazolli, "Massively parallel tree search for high-dimensional sphere decoders," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2309–2325, 2018.

[7] M. T. Nguyen, X. N. Tran, V. D. Ngo, Q.-K. Trinh, D. T. Nguyen, and T. A. Vu, "Sub-optimal deep pipelined implementation of MIMO sphere detector on FPGA," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 10, no. 1, pp. e3–e3, 2023.

[8] E. Grell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Transactions on Information Theory*, 2002.

[9] H. Vikalo, B. Hassibi, and T. Kailath, "Iterative decoding for MIMO channels via modified sphere decoding," *IEEE Transactions on Wireless communications*, vol. 3, no. 6, pp. 2299–2311, 2004.

[10] L. Brunel, "Multiuser detection techniques using maximum likelihood sphere decoding in multicarrier CDMA systems," *IEEE Transactions on Wireless Communications*, vol. 3, no. 3, pp. 949–957, 2004.

[11] K. Nikitopoulos, "Massively parallel, nonlinear processing for 6G: Potential gains and further research challenges," *IEEE Communications Magazine*, vol. 60, no. 1, pp. 81–87, 2022.

[12] M.-A. Arfaoui, H. Ltaief, Z. Rezki, M.-S. Alouini, and D. Keyes, "Efficient sphere detector algorithm for massive MIMO using GPU hardware accelerator," *Procedia Computer Science*, vol. 80, pp. 2169–2180, 2016.

[13] G. Georgis, K. Nikitopoulos, and K. Jamieson, "Geosphere: An exact depth-first sphere decoder architecture scalable to very dense constellations," *IEEE Access*, vol. 5, pp. 4233–4249, 2017.

[14] M. W. Hassan, A. Dabah, H. Ltaief, and S. A. Fahmy, "Signal detection for large MIMO systems using sphere decoding on FPGAs," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2023.

[15] M. Wu, B. Yin, G. Wang, C. Studer, and J. R. Cavallaro, "GPU acceleration of a configurable n-way MIMO detector for wireless systems," *Journal of Signal Processing Systems*, vol. 76, pp. 95–108, 2014.

[16] L. G. Barbero and J. S. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," in *IEEE Workshop on Signal Processing Advances in Wireless Communications*, 2006.

[17] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of solid-state circuits*, vol. 40, no. 7, pp. 1566–1577, 2005.

[18] A. Burg, S. Haene, D. Perels, P. Luethi, N. Felber, and W. Fichtner, "Algorithm and VLSI architecture for linear MMSE detection in MIMO-OFDM systems," in *IEEE International Symposium on Circuits and Systems*, 2006.

[19] M. Li, B. Bougard, E. E. Lopez, A. Bourdoux, D. Novo, L. Van Der Perre, and F. Catthoor, "Selective spanning with fast enumeration: A near maximum-likelihood MIMO detector designed for parallel programmable baseband architectures," in *IEEE International Conference on Communications*, 2008.

[20] H. Wang and M. Chen, "A fixed-complexity sphere decoder for MIMO systems on graphics processing units," in *International Conference on Information Engineering and Computer Science*, 2010.

[21] Y. Meng, R. Kannan, and V. Prasanna, "Accelerating monte-carlo tree search on CPU-FPGA heterogeneous platform," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2022.

[22] Y. Wu and J. McAllister, "Configurable quasi-optimal sphere decoding for scalable MIMO communications," *IEEE Transactions on Circuits and Systems I*, vol. 68, no. 6, pp. 2675–2687, 2021.

[23] K. Nikitopoulos, J. Zhou, B. Congdon, and K. Jamieson, "Geosphere: Consistently turning MIMO capacity into throughput," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 631–642, 2014.

[24] T.-H. Kim and I.-C. Park, "Implementation of a high-throughput and area-efficient MIMO detector based on modified Dijkstra's search," in *GLOBECOM IEEE Global Telecommunications Conference*, 2009.

[25] J. Chen, C. Fei, H. Lu, G. E. Sobelman, and J. Hu, "Hardware efficient massive MIMO detector based on the monte carlo tree search method," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 4, pp. 523–533, 2017.

[26] J. Chen, S. Chen, Y. Qi, and S. Fu, "Intelligent massive MIMO antenna selection using monte carlo tree search," *IEEE Transactions on Signal Processing*, vol. 67, no. 20, pp. 5380–5390, 2019.

[27] Xilinx. Alveo U280 data center accelerator card data sheet. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds963-u280.pdf