Signal Detection for Large MIMO Systems Using Sphere Decoding on FPGAs

Mohamed W. Hassan, Adel Dabah, Hatem Ltaief, Suhaib A. Fahmy
King Abdullah University of Science and Technology (KAUST)
Extreme Computing Research Center (ECRC)
Thuwal, Saudi Arabia
{mohamed.hassan,adel.dabah.1,hatem.ltaief,suhaib.fahmy}@kaust.edu.sa

***Abstract*—Wireless communication systems rely on aggressive spatial multiplexing Multiple-Input Multiple-Output (MIMO) access points to enhance network throughput. A significant computational hurdle for large MIMO systems is signal detection and decoding, which has exponentially increasing computational complexity as the number of antennas increases. Hence, the feasibility of large MIMO systems depends on suitable implementations of signal decoding schemes.**

This paper presents an FPGA-based Sphere Decoder (SD) architecture that provides high-performance signal decoding for large MIMO systems, supporting up to 16-QAM modulation. The SD algorithm is refactored to map well to the FPGA architecture using a GEMM-based approach to exploit the parallel computational power of FPGAs. We implement FPGA-specific optimization techniques to improve computational complexity. We show significant improvement in time to decode the received signal with under 10^{-2} BER. The design is deployed on a Xilinx Alveo U280 FPGA and shows up to a 9$\times$ speedup compared to optimized multi-core CPU execution, achieving real-time requirements. Our proposed design reduces power consumption by a geo-mean of 38.1$\times$ compared to CPU implementation, which is important in real-world deployments. We also evaluate our design against alternative approaches on GPU.

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

I. Introduction

Modern and emerging applications are increasingly reliant on high-performance networks, covering diverse use cases such as video conferencing, Internet of Things (IoT) offload, and collaborative automotive applications. Such growing demand has pushed wireless network technology to enable higher capacity and throughput requirements without breaking the constraint of real-time response (we assume the real-time response to be within $10ms$ [1]). Multiple-Input Multiple-Output (MIMO) systems are an established approach to address increased demand, and it features in cellular and local-area networking standards [2, 3]. Detection algorithms are one of the main hurdles in realizing such aggressive spatial multiplexing systems, where there is a trade-off in performance (i.e., in terms of accuracy) and complexity (i.e., in terms of time to solution) between utilizing linear and non-linear detection schemes [1, 4–7].

Linear decoders, such as {Maximum Ratio Combining (MRC), Zero Forcing (ZF), and Minimum Mean Square Error (MMSE)}, are characterized by low complexity but have poor Bit Error Rate (BER) performance [1, 8, 9]. On the other hand, non-linear decoders exhibit good BER performance but are computationally intensive. The complexity of non-linear decoders is exponential, which makes it challenging to achieve real-time processing when scaling the number of antennas in MIMO systems.

The Maximum Likelihood (ML) [10] decoder is considered to be the optimal non-linear decoding algorithm; however, it incurs very high complexity [1, 8]. The Sphere Decoder (SD) algorithm [11, 12] is a variant of the ML decoder that aims to reduce its complexity by restricting the search space to process selected candidate nodes efficiently. To achieve this, most SD methods use heuristics to identify promising candidate tree nodes during exploration, which may subsequently impact the accuracy of the detection algorithm [4, 13].

While the SD algorithm reduces the computational complexity of signal decoding, it still scales exponentially, significantly limiting the scalability of MIMO systems. Consequently, the wireless communications community is currently assessing the application of hardware accelerators to reduce signal decoding time. High-Performance Computing (HPC) architectures, such as GPUs, provide significant computational power. However, computational time cannot be the only focus of optimization; power efficiency must also be considered since such signal detection systems are usually deployed in remote base stations with a limited power budget. Furthermore, the overall application latency, including the accelerator, must be considered. Hence, FPGAs are a promising solution for addressing this trade-off by providing power-efficient, high-performance designs with flexible interconnectivity. Moreover, FPGA boards have high-speed transceivers that are very beneficial when deploying MIMO systems in a real-world setup. While CPUs and GPUs would require data to be streamed into memory and then accessed by the computational device, FPGAs have the hardware capability to stream data directly into the computational pipeline, bypassing the memory access step.

This paper presents an FPGA-based signal detection accelerator using the SD algorithm. We adopt a GEMM-based variant of the SD algorithm proposed in [1] that refactors the algorithm from memory-bound to compute-bound. We combine this approach with the Best-First Search tree traversal strategy proposed in Geoshpere [14] to mitigate the complexity of the Breadth First Search (BFS) tree traversal in [1]. The

approach proposed in [1] achieves *almost* real-time decoding with acceptable BER performance at a Signal-to-Noise Ratio (SNR) of 17 dB by utilizing GPU acceleration. However, our approach exploits FPGA data- and pipeline-parallelism to reach comparable BER performance at an SNR of only 4 dB while showing significant improvement in the decoding time.

Direct porting of the SD algorithm to the FPGA shows comparable performance to CPU execution using the vendor-optimized Math Kernel Library (MKL). Hence, we apply FPGA-specific optimizations to enhance decoding time. Compared to an optimized multi-core CPU implementation, our optimized FPGA design results in up to 9× speedup. The contributions of this work are as follows:

- We devise a hardware-oriented design for the SD algorithm that maps well to FPGAs with hardware optimizations that improve compute complexity without impacting BER performance (Section III).
- We demonstrate that our designs outperform an optimized CPU implementation allowing the accommodation of larger MIMO systems without exceeding real-time constraints (Section IV).
- We demonstrate considerable energy savings, compared to the optimized CPU implementation (Section IV-B).
- We show that the Best-First Search strategy adopted in our design significantly reduces the search space, significantly accelerating the decoding process compared to an alternative GPU implementation based on heuristics (Section IV-F).

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 the additive white Gaussian noise, where n_i is an independent zero-mean circularly symmetric complex Gaussian random variable with variance σ^2.

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

The ML decoder, which is considered to be the optimal non-linear decoding algorithm, calculates a posterior probability for all possible transmitted vectors $s \in S$, where $|S| = |\Omega|^M$, as shown in Equation 2. The ML decoder returns the vector s that minimizes the distance between the received vector y and the assumed noiseless vector Hs. While the ML decoder explores all possible states, the SD algorithm limits the search space within a radius to reduce complexity.

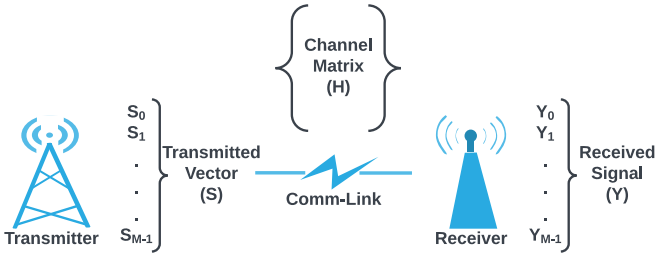

Fig. 1: Typical M×N MIMO system.

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

B. Sphere Decoding (SD)

The SD algorithm solves Equation 2 by enumerating the points inside a hyper-sphere 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.

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

The SD algorithm represents the problem as a search tree that enumerates all possible combinations of the transmitted vector. The algorithm traverses the search tree to identify the best path minimizing the distance from the received signal y. Tree nodes evaluations are compared with the pre-set radius r to identify and prune non-promising tree branches, thereby reducing the computational complexity. Figure 2 illustrates the tree pruning process, where the value inside each node represents the Partial Euclidean Distance (PD) of the node. The tree represents the search space for three transmitters with BPSK modulation and a sphere radius $r = 10$. When the PD evaluation of a node exceeds the preset radius, the branch is cut under the assumption that it yields non-promising leaf nodes, thereby reducing the search space. The nodes highlighted in red show the path to the optimal solution.

We adopt a variant of the SD algorithm proposed in [1], which is a GEMM-based approach. This refactors the SD algorithm from memory-bound to compute-bound to be able to exploit the aggressive computing capability of hardware accelerators like GPUs and FPGAs. This transformation is achieved by casting memory-bound matrix-vector multiplication (i.e., Level-2 BLAS) to matrix-matrix multiplication operations (i.e., Level-3 BLAS). The optimization problem shown in Equation 3 is translated to 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 and $R \in C^{N \times M}$ is an upper triangular matrix. This translation transforms Equation 3 to be represented as shown in Equation 4:

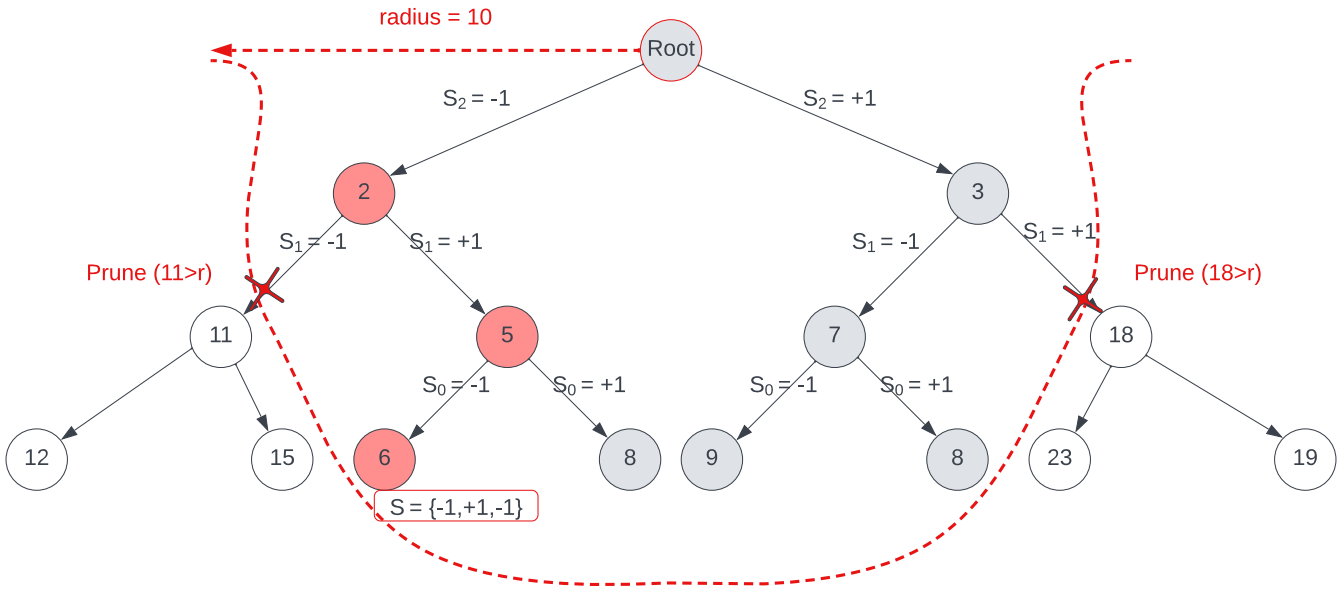

Fig. 2: Sphere Decoding example tree showing the search space for three transmit antennas with BPSK modulation and radius $r = 10$. Where, S_i represents the decoded symbol value for that edge, and the number inside each node is the Partial Distance (PD) evaluation of the node. The final vector S at leaf nodes represents the entire decoded signal.

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

where $\bar{y} = Q^H y$. Hence, the supposed transmitted vector $\hat{s}$ is evaluated by solving the minimization problem shown in Equation 5 and elaborated in Equation 6 as follows.

$$\min \sum_{k=1}^{M} g_k(s_{M-1}, ..., s_{M-k}) \tag{5}$$

$$g_k(s_{M-1}, ..., s_{M-k}) = ||\bar{y}_{M-k} - \sum_{i=M-k}^{M-1} (r_{(M-k),i} s_i)||^2 \tag{6}$$

The computational profile of the SD algorithm includes three main processes: branching, evaluation, and pruning, as shown in Algorithm 1. The branching process generates all possible successors of a tree node, which models enumerating all possible states of one symbol in the transmitted vector. Then, the evaluation process computes the Partial Distance (PD) of the generated tree nodes, which estimates the distance of these nodes from the received signal. Finally, the pruning process uses the evaluated PDs of the generated nodes to check if they reside within the sphere radius r to cut unpromising branches.

The tree exploration strategy can be Breadth First Search (BFS), Depth First Search (DFS), or Best First Search (Best-FS). While BFS exposes more parallelism, DFS and Best-FS

Algorithm 1: SD Algorithm

Data:
Received signal y
Constellation order Ω
Channel matrix estimation H
Noise variance σ^2
Radius r
Result:
Decoded signal vector $\hat{s}$

$List \leftarrow root$;
while $List \neq 0$ **do**
  $Node = pop(List)$;
  Branch($Node$) ; /* Generate $Node_{i:1 \rightarrow |\Omega|}$ */
  foreach $Node_i$ **do**
    if $E(Node_i) < r$ **then**
      /* $E()$ computes Node PD */
      if $Node_i \equiv$ *Leaf node* **then**
        $r = E(Node_i)$;
        /* Update sphere radius */
        $\hat{s} = Symbols(Node_i)$;
        /* Best solution so far */
      else
        $List = List \cup Node_i$;
      end
    else
      Prune branch;
    end
  end
end

explore leaf nodes first. This paper focuses on the leaf-node biased exploration strategies for faster sphere radius update, which helps prune the search space more efficiently than BFS. More specifically, we target Best-FS exploration as it targets better quality leaf nodes (in terms of PD) by sorting children nodes generated at each tree level. The sorting overhead is negligible since its complexity depends only on the modulation parameter and is dominated by the GEMM complexity.

C. Related Work

Monte-Carlo Tree Search (MCTS) has been successful in decision-making problems and has been applied to MIMO signal detection by Chen et al. in [13], then later for antenna selection in [15]. The MCTS-based MIMO system proposed in [13] maps MIMO detection to MCTS and implements it in hardware. Their proposed approach eliminates the need for costly matrix inversion or Gram matrix computation. Multiple optimization techniques are utilized to enhance the decoding process's complexity and performance. They limit the search space using a pre-value network, reduce signal interference using a level re-order method, and optimize BER using a multi-search process. The design is implemented on 65 nm CMOS for a 64×8 MIMO detector achieving 10^{-6} BER at SNR of -0.4 dB with QPSK and 2.75 dB with 16-QAM. In other efforts to limit the search space, a statistical tree pruning approach was proposed in [16] to eliminate non-promising tree branches. While their heuristic shows good BER performance, their approach does not satisfy the real-time constraint successfully.

In an attempt to optimize communication between parallel workers of MCTS, [17] proposes the decomposition of the MCTS decision tree into two separate data structures, State Table (ST) and Upper Confidence-bounded Tree (UCT). While UCT is mapped to an optimized FPGA pipeline for in-tree computations, the ST keeps an index for each node in the tree. ST is mapped to the CPU to keep the search tree's state information. Parallel send and receive buffers are implemented to optimize data flow between parallel CPU threads and the FPGA's pipeline. This approach proved scalable for many parallel worker threads and achieved up to 3× throughput compared to state-of-the-art CPU-based parallel MCTS solvers. However, this was applied to more traditional MCTS applications.

Nikitopoulos et al. [4] propose a VLSI architecture that massively parallelizes the search tree of sphere decoders while keeping complexity close to that of the optimal sequential decoders. They employ a multi-sphere design introducing the "tree of promise" concept where symbols are described by their relative ordered distance to the received signal. They partition the tree of promise and process it in parallel; however, the partitioning overhead, which scales only linearly, is considered to be an offline computation. While the sub-trees are nearly data-independent, they still require a synchronization step once they reach the first leaf node to communicate the minimum euclidean distance to all parallel sub-trees. They achieve a 29× reduction in latency for a 16-QAM 10×10 MIMO system using 32 processing elements compared to sequential SD. An alternative approach was proposed by Husmann et al. in [18], antipodal detection and decoding mechanism. The premise of this work is that the antipodal detector either results in a reliable solution or non at all (erasure) to skip the complexity of the workload if a promising solution is unlikely. They propose a belief propagation-based decoder that deals with the erasure and attempts to reconstruct a correct signal.

While the workload of SD can be embarrassingly parallel at each level of the search tree, there is a necessary sorting operation at the end of each level's computation that imposes data dependency for the following data set. Hence, scalability is challenging to achieve for such an algorithm. The fixed complexity SD (FSD) sacrifices ML optimality to enhance scalability. FSD is massively parallelizable with minimal dependencies among the parallel tasks [5]; however, this approach is very resource hungry, where it requires the available number of PEs to be a multiple of the transmitted constellations [9].

Other works combining both GPU and FPGA accelerators are proposed in [19]. Their approach includes a pre-processing phase that prepares the SD search tree, then offloads it to either the GPU or the FPGA. The work presented in this paper eliminates the communication overhead by building the search tree directly in the FPGA's hardware.

The work proposed in [1] uses a GEMM-based variant of the SD algorithm on GPUs. However, they rely on the BFS search strategy to enhance the GPU hardware utilization but ultimately increase the overall complexity. On the other hand, Geosphere [14] is a sphere decoding-based system that uses geometric reasoning on the signal constellation to define how to traverse the search tree. Geosphere utilizes the Best First Search approach, as previously explained in Section II-B. This paper presents an approach combining the strengths of both Geosphere [14] and the GEMM-based approach [1] to overcome the MIMO complexity challenge.

III. Sphere Decoder Implementation on FPGA

This section details the implementation of the SD algorithm on FPGA. The scope of this work targets large MIMO systems that have proven to be prohibitive in terms of complexity when using multi-core CPUs. Hence, we discuss the implementation of a hardware-accelerated SD algorithm and the FPGA optimizations employed to reduce the time to solution.

We start by profiling the SD algorithm to identify bottlenecks and understand the computational requirements for each phase of the execution flow. We discuss details of the execution profile and elaborate on its mapping to the FPGA's pipeline.

A. SD Execution Profile

The search tree that models the SD algorithm includes M levels, where each level corresponds to a constellation symbol of its respective transmitter. The tree is built dynamically as the algorithm progresses in search of the decoded signal. Hence, the tree's root node is initialized with all symbols in the vector $\hat{s}$ unknown.

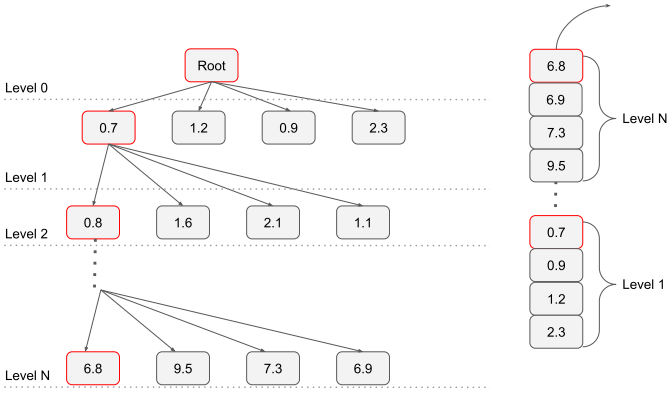

Fig. 3: Tree expansion illustration with visualization of sorted node insertion in the tree list data structure. Nodes are popped in a Last In First Out (LIFO) sequence.

1) Phase 1 (Branching): The branching process generates all possible successors to the tree node being processed. Child nodes at level L inherit the parent's known symbols and enumerate all possibilities of the symbol $\hat{s_L}$. A matrix is then constructed to model the tree's current state at each node.

2) Phase 2 (Evaluation): The evaluation of each node's Partial Distance (PD) includes a matrix multiplication followed by a norm computation. For each node in the tree, a block of the tree state matrix is multiplied by its corresponding block in the channel matrix. The resulting matrix is normalized with the received signal to calculate the node's PD.

3) Phase 3 (Pruning): This phase has a significant impact on the complexity of the SD algorithm. During this phase, the algorithm determines which branches to prune to reduce the search space. Hence the sorting process plays an important role, where the nodes at the current level being evaluated are sorted based on their PDs. The algorithm favors branches with lower PDs since they are more likely to lead to a more promising solution. Sorting is conducted at the end of each level generation to prioritize lower PD nodes, as shown in Figure 3.

The computational profile of the SD algorithm is sequential in nature, as it requires a synchronization step to update the sphere radius after reaching a leaf node. While this synchronization step is a performance hurdle, it significantly reduces the time to solution via efficient pruning of the tree search space. The criteria upon which a branch is cut depends on the sphere radius. Figure 3 shows an example of SD algorithm progression in the search tree. The value inside each node represents its Partial Distance (PD). The data structure on the right of the figure is a visualization of the tree list data structure and explains how nodes are inserted in order. For illustration purposes, the numbers inside the node represent the PD evaluation of each node; however, in the actual implementation, this data structure keeps node IDs that link to an entry in a separate data structure, the Meta State Table (MST).

B. FPGA Pipeline Overview

Here, we discuss mapping the SD algorithm to a pipelined data-flow architecture implemented on the FPGA. An abstract view of the pipeline is shown in Figure 4. Initially, data is transferred once through PCIe from the host to the FPGA's High Bandwidth Memory (HBM). This is a one-time negligible overhead that is evaluated empirically to be less than 3% of the overall execution time. The search tree construction is fully implemented in the FPGA fabric to avoid repeated communication with the host. This design choice integrates well with real-word deployment by utilizing the high bandwidth of the FPGA's streaming I/O interfaces.

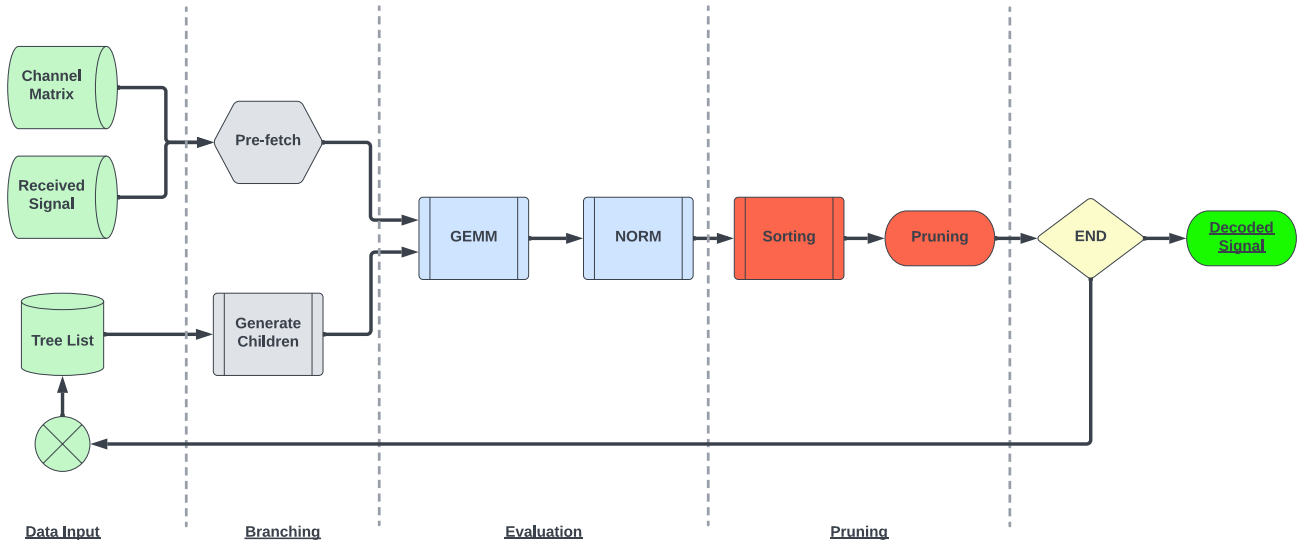

Fig. 4: Dataflow pipeline implemented on the FPGA, detailing the building blocks of the hardware accelerator.

The branching phase modules exploit pipeline parallelism to generate P work tasks, where P is the modulation factor of the MIMO system and corresponds to the number of children generated by each node. A tree state matrix is constructed/updated with the generated node information and stored in the on-chip dual-port Block RAM (BRAM). The BRAM storage allows fast single-cycle memory access for this intermediate data during phase 2 of the pipeline, the evaluation.

The evaluation phase combines data and pipeline parallelism to efficiently process the evaluation of the generated nodes at a specific level in the tree. An optimized GEMM engine is used to carry out the large number of matrix multiply operations and feeds the results to the normalization module (NORM) for final PD calculation. Partitioned arrays are used to store the intermediate data of the generated branches on-chip, pending a decision to commit from the pruning module.

C. FPGA-Specific Optimizations

Three main challenges were identified when mapping this problem to an FPGA pipeline: the large number of GEMM floating-point operations, the irregularity of the memory accesses, and the dynamic nature of the problem. These challenges require addressing if the performance of the FPGA accelerator is to be maximized [20, 21]. Here, we outline the techniques we used to yield an efficient FPGA design.

1) Optimized GEMM Engine: Not only is floating-point matrix multiplication itself a compute-intensive operation, but it is executed no less than $\frac{P^{N+1}-1}{P-1}$ in the worst-case scenario, where P is the modulation factor, and N is the number of receiver antennas. Observing the execution profile showed that this operation is a significant bottleneck; hence we address this challenge by implementing a highly optimized GEMM engine. The implementation is adapted from the Xilinx BLAS library [22], which includes BLAS engines that implement memory and control logic to compose complex BLAS functions. These present an overhead for our specific application since we only require the GEMM operation. Hence, we extract and isolate the GEMM engine to make it efficient.

The GEMM engine is constructed as a systolic array that utilizes floating-point MAC units built using the FPGA's DSP slices. A two-dimensional mesh of MAC units is augmented with control logic and connected to single-cycle access BRAM memory blocks. This configuration ensures smooth matrix multiply computation minimizing bubbles in the architecture's pipeline.

2) Pre-Fetching Unit: Inputs to the GEMM engine include the channel matrix, the received signal, and the tree state matrix. However, only partial blocks of these matrices are used for computation. Which data should be extracted from these large matrices depends on the node currently being processed and its level in the search tree. Moreover, accessing these blocks does not have a regular pattern. A fixed memory access stride can be defined if the whole search tree needs to be explored. However, pruning the search space is one of the critical features of the SD algorithm, which leads to an unpredictable traversal path of the search space. Hence, this portion of the computation's memory access pattern becomes irregular.

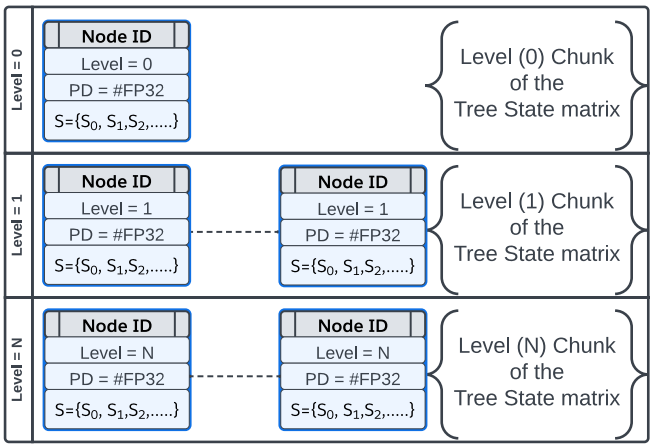

Fig. 5: Visualization of the Meta State Table (MST). It represents the database used to store the tree state data, which is partitioned corresponding to each level's data.

We implemented a pre-fetching unit that pre-calculates the required memory addresses needed by the GEMM engine via level and node information. Then, the pre-fetching unit extracts the required data and buffers it contiguously to be easily offloaded to the GEMM engine Block RAMs. This double buffering approach eliminates the high latency memory accesses that the matrix multiply operation would have imposed.

3) Meta State Table (MST): The search tree is built dynamically in the FPGA's re-configurable fabric to avoid communication with the host. Moreover, the data structures responsible for storing the tree state information are dynamic in size. Dynamic data structures are not supported on the FPGA and pointer-to-pointer addressing is very inefficient in terms of performance. Hence, we implement a Meta State Table (MST). This is a database that keeps track of the current status of the search tree. We use it to index the tree nodes to overcome the challenge of dynamic tree construction. Furthermore, it decouples pointer-based addressing by keeping an updated copy of the tree state matrix.

The tree state matrix is a matrix that models the state of the tree based on the recovered symbols of a node's parents tracing its branch back to the root node. Instead of storing the whole matrix in one large data structure, the MST keeps a record of a node's corresponding block in the tree state matrix, as shown in Figure 5. Such a node-oriented data structure facilitates partitioning the memory allowing single-cycle access to the data. Moreover, it eliminates pointer-based addressing, significantly impacting the FPGA design's performance.

4) Resource Utilization Optimization: The resource utilization of the generated hardware is a very important aspect of the design. If the synthesized pipeline consumes more than 50% of the available resources on the FPGA, one may not be able to instantiate a second pipeline path to exploit

TABLE I: FPGA resource utilization

	Baseline 4-QAM	Baseline 16-QAM	Optimized 4-QAM	Optimized 16-QAM
Freq (MHz)	253	253	300	300
LUTs	29%	50%	11%	23%
FFs	20%	27%	7%	11%
DSPs	8%	15%	3%	7%
BRAMs	11%	14%	8%	10%
URAMs	14%	60%	7%	30%

TABLE II: Power profile for CPU and FPGA

		10×10 4-QAM	15×15 4-QAM	20×20 4-QAM	10×10 16-QAM
Power (W)	**CPU**	82	93	135	142
	FPGA	8	11.7	12	12.8
Exec (ms)	**CPU**	7	44.3	350.6	176.6
	FPGA	2	9.4	102.5	46.88
Energy (J)	**CPU**	0.574	4.11	47.3	25.1
	FPGA	0.016	0.11	1.23	0.6
Energy Reduction		35.8×	36.8×	38.4×	41.8×

more data parallelism. Hence, resource utilization is a very important aspect to consider when designing the hardware. To allow room for parallelization in the future, we optimize the hardware design to reduce resource utilization.

We simplify the control logic by implementing a separate design for each modulation configuration (e.g., 4-QAM or 16-QAM). This helps eliminate much of the control logic responsible for sequencing the operations. Moreover, we employ buffer reuse techniques to reduce the on-chip memory usage of BRAM and URAM memory blocks. These techniques help save a significant amount of resources that may enable future work on exploring data parallelism to enhance the performance of the SD algorithm further.

IV. Experimental Results

A. Experimental Setup

The experimental platform used in this work includes a Xilinx Alveo U280 FPGA card hosted in a workstation with an AMD Ryzen CPU via PCIe. The Alveo U280 is equipped with 8GB of High Bandwidth Memory (HBM) accessible over 32 channels in addition to 32 GB of DDR4 memory [23]. It also includes 4032 Block RAMs (18Kb each) and 960 URAM blocks (288Kb each). The FPGA designs are implemented using OpenCL/C++ High-Level Synthesis (HLS) and synthesized using Vitis 2020.2, with the designs running at an approximate frequency of 300 MHz. The 64-core CPU runs Ubuntu 18.04. We use Intel Math Kernel Library (MKL) in association with the Boost library to implement the optimized CPU version of the SD algorithm. The testing data set is randomly generated using Monte Carlo simulations to emulate

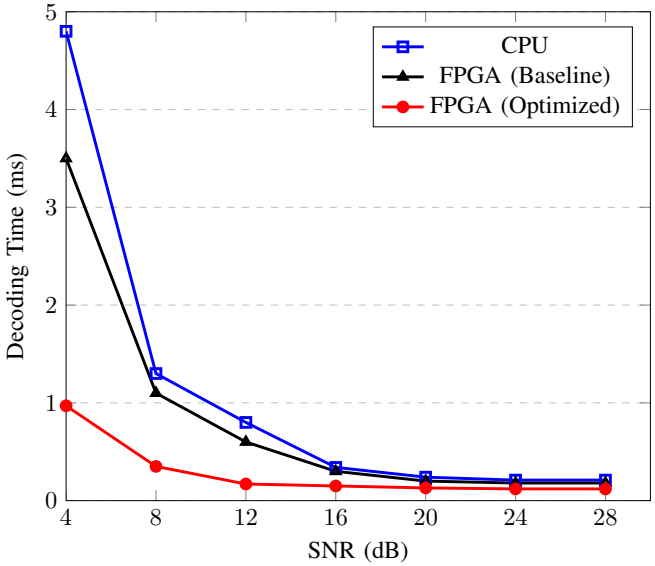

Fig. 6: Execution time for 10×10 MIMO with 4-QAM modulation.

the MIMO system. While 4×4 MIMO is a realistic current test case, we scale up testing to a 20×20 MIMO configuration.

B. Power Profile and Resource Utilization

Due to the limited resources on the FPGA board, resource utilization is usually a concern. We show the resource utilization for our implemented designs in Table I. The baseline implementation is a direct port of the SD algorithm to the FPGA. The utilized portion of Look Up Tables (LUTs) and Ultra RAMs (URAM) in the baseline implementation prevent the instantiating of multiple pipelines. However, applying the optimization techniques explained in Section III shows a significant reduction in resource utilization.

We performed power profiling for the CPU using AMDuprof and for the FPGA using Vitis Analyzer. The power consumed during signal decoding for all the design configurations is detailed in Table II. The energy consumed by each test case shows a significant advantage to the FPGA implementation, reducing the energy consumption by a geomean of 38.1× compared to the CPU implementation.

C. Performance Analysis

The baseline FPGA implementation is a direct port of the SD algorithm's C++ code compiled using Xilinx HLS tools (After eliminating dynamic memory structures and global variable declarations). As shown in Figure 6, the baseline implementation shows comparable performance to the CPU with a slight advantage for the FPGA. Around 1.4× speedup is observed at an SNR of 4 dB for a 10×10 MIMO configuration with 4-QAM modulation. While for that MIMO configuration, all the implementations (CPU, FPGA-baseline, FPGA-optimized) satisfy the real-time constraint requirement ($\leq 10ms$), the optimized FPGA implementation shows a significant speedup of 5× compared to CPU execution at an SNR of 4 dB.

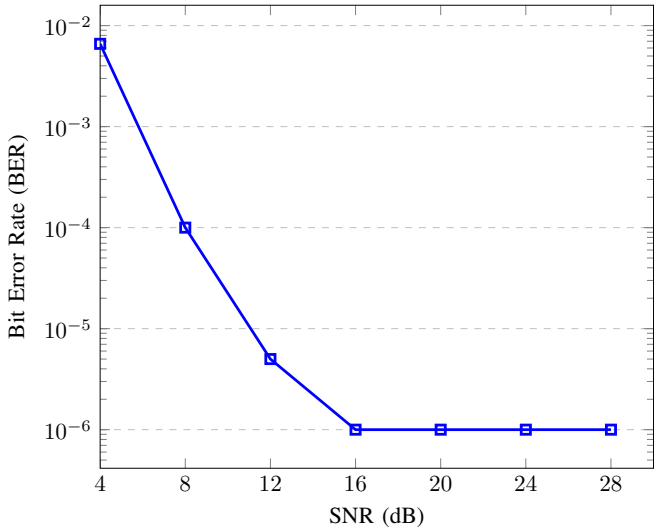

Fig. 7: Bit Error Rate (BER) for 10×10 MIMO with 4-QAM modulation.

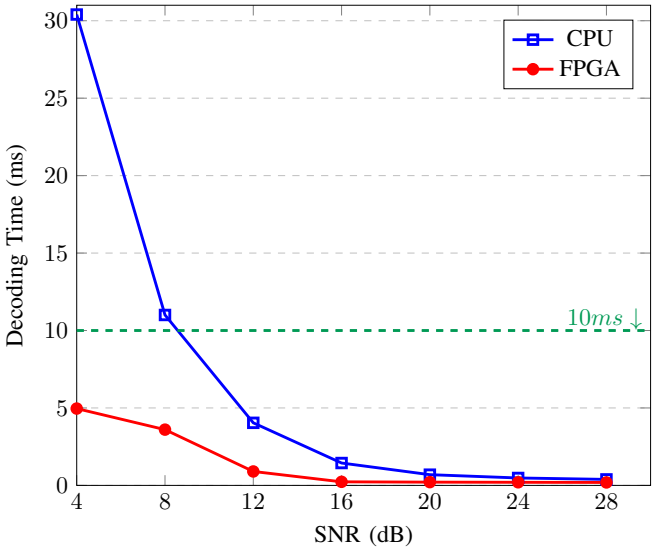

Fig. 8: Execution time for 15×15 MIMO with 4-QAM modulation.

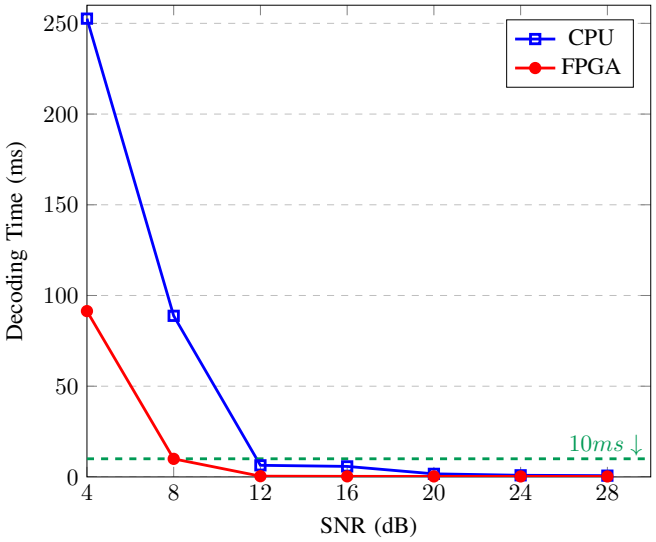

Fig. 9: Execution time for 20×20 MIMO with 4-QAM modulation.

In designing the hardware architecture, we are careful to mimic the execution profile and operational sequence of the CPU execution. Hence, the employed hardware optimizations enhance the design's throughput without affecting BER performance. For the 4-QAM 10×10 MIMO configuration, the BER performance is below 10^{-2} even for the lowest tested SNR of 4 dB, as shown in Figure 7.

D. Scaling the Number of Antennas

The SD algorithm's complexity is exponential; increasing the number of antennas has a significant impact on the time to decode the signal. Figure 8 shows that the CPU implementation breaks the real-time constraint when the number of antennas increases to 15×15. At an SNR of 4 dB, the CPU implementation decodes the signal in over $30ms$ breaking real-time requirements. It starts reaching almost real-time decoding at an SNR of 8 dB, while on the other hand, the FPGA optimized implementation shows a 6.1× speedup reducing the decoding time to $5ms$, satisfying the real-time constraint, as demonstrated in Figure 8.

A larger MIMO system of size 20×20 incurs a high decoding time at an SNR of 4 dB in both CPU and FPGA implementations. Nevertheless, the FPGA implementation successfully decodes the received signal in $9.9ms$ at an SNR of 8 dB, as shown in Figure 9, while CPU decoding took $88.8ms$. The 9× speedup of the FPGA implementation compared to the CPU makes it possible to decode the signal in real-time.

E. Scaling the Modulation Factor

16-QAM modulation is significantly more complex to decode than 4-QAM due to the larger number of child nodes for each expansion of a tree node. Figure 10 shows that the decoding time for a 10×10 MIMO system takes almost $100ms$ on the CPU at an SNR of 4 dB. The CPU implementation only starts meeting the real-time constraint at an SNR of between 16 dB and 20 dB. The FPGA design is 4× faster than the CPU, almost achieving real-time decoding at an SNR of 8 dB. The increase of the modulation factor affects the complexity of the SD algorithm more than scaling the number of antennas.

The culprit for this considerable increase in complexity is the size of the intermediate tree state matrix. The size of this matrix is in the order of $(4 \times Modulation^2 \times N)$, where N is the number of receive antennas. Hence, for the 4-QAM configuration, the size of this matrix is multiplied by a factor of $(4^2) = 16$, while for 16-QAM, the multiplying factor is $(16^2) = 256$. The tree state matrix is used in the GEMM computation involved in evaluating the PD of each node in the tree. Since this computation occurs many times during the decoding process, increasing its complexity will have a significant impact on the overall decoding time.

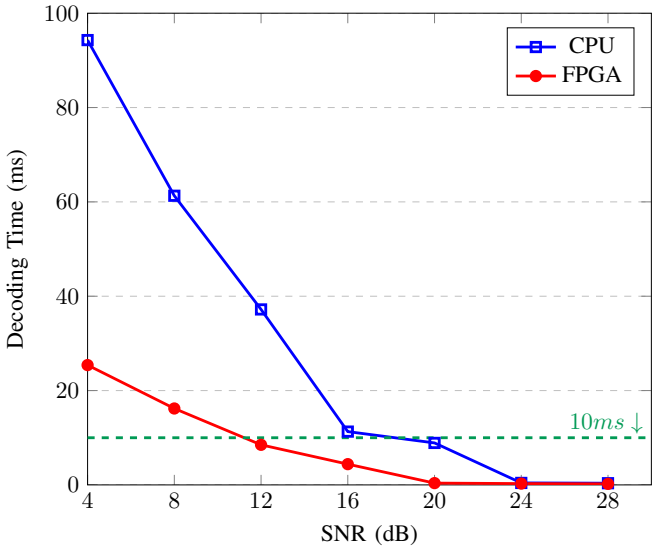

Fig. 10: Execution time for 10×10 MIMO with 16-QAM modulation.

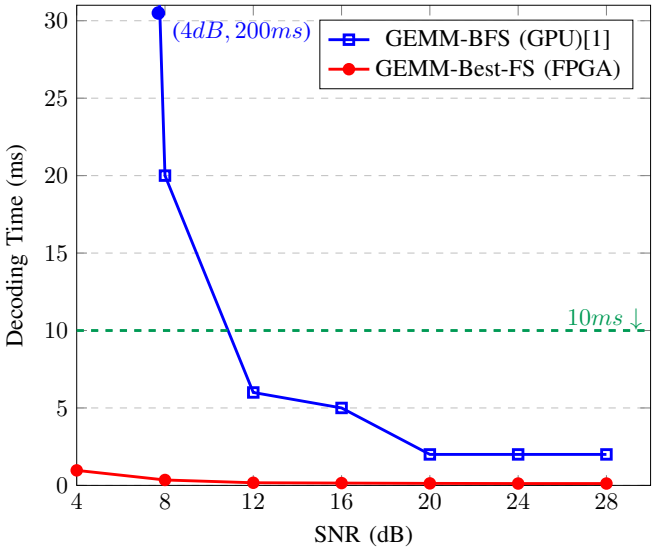

Fig. 11: Execution time for 10×10 MIMO with 4-QAM modulation.

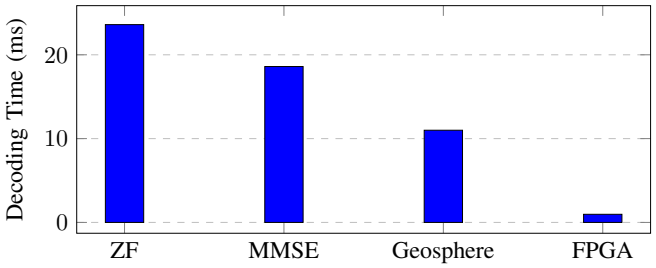

Fig. 12: Decoding time comparison for 10×10 MIMO.

F. Performance Discussion

The GPU-based implementations [1, 19, 24] attempt to utilize the massive computational power of the GPU cores to decode the received signal. However, the SD algorithm includes a synchronization operation with every global time step. The synchronization operation is responsible for updating the sphere radius at runtime to aid in the tree-pruning process, which significantly reduces the search space. However, such a synchronization step is very costly on GPUs. Hence, most GPU optimization strategies depend on heuristics to limit the search space, but this comes at the cost of BER performance. In our work, we focus on the optimal SD algorithm to avoid any BER performance degradation.

The work presented in [1] uses a GEMM-based variant of the SD algorithm with a BFS exploration strategy. Here, we elaborate on a performance comparison with the GEMM-BFS approach in [1] reproduced on an Nvidia A100 GPU for the specific configurations in this work. Our FPGA optimized design shows an average speedup of 57× compared to the GPU implementation in [1] as illustrated in Figure 11. For a 4-QAM 10×10 MIMO configuration, the GPU implementation decoded the signal in 6 ms at an SNR of 12 dB. On the other hand, our FPGA implementation decoded the signal for the same configuration in 0.97 ms at an SNR of 4 dB. The main reason for the significant speedup of this work is incorporating the DFS approach in our design versus using BFS in the GPU implementation of [1]. A BFS search strategy exposes dependence-free parallelism that has the potential to efficiently utilize the GPU's cores. However, such an exploration strategy would reach the solution leaf nodes after exploring the whole tree. On the other hand, the DFS combined with the sorting search strategy (implemented in our approach) prunes the search space to less than 1% of the number of explored nodes. Consequently, there is a considerable speedup when compared to the approach in [1]. Moreover, The FPGA's architecture combined with a high-performance GEMM engine significantly improves the signal decoding time. Hence, our design decodes the signal in real-time at far lower SNR values than other approaches.

Figure 12 shows a performance comparison for a 4-QAM 10×10 MIMO system between our approach (FPGA-optimized), ZF, MMSE, and Geosphere [14], implemented on a Rice WARP v3 radio programmable platform [25]. We show significant improvement in the decoding time as well as savings in the SNR values. Geosphere [14] decodes the signal in 11 ms at an SNR of 20 dB; we show 11× speedup in the decoding time with a reduction in the SNR to 4 dB. Augmenting Geosphere's Best First Search tree traversal strategy with the GEMM-based approach eliminates the memory-bound aspect of the execution profile. Moreover, it maps well to the FPGA's architecture.

V. Conclusion

MIMO systems are a key technology enabling high-performance wireless networks. However, signal decoding is a significant bottleneck that hinders the scalability of such systems with a large number of antennas. Linear decoders are computationally attractive, but they exhibit poor BER

performance, especially when scaling the number of antennas. On the other hand, non-linear decoders provide good BER performance but are computationally complex. Hence, this paper tackles the challenging problem of non-linear decoding algorithms to achieve better scalability.

We focused on the Sphere Decoding (SD) algorithm, which is a variant of the Maximum Likelihood (ML) non-linear decoding algorithm. We employed a refactored version of the SD algorithm that transforms the memory-bound computation to a GEMM-based compute-bound operation. We used an optimized GEMM engine composed of a systolic array combined with a double buffering technique to execute fast matrix multiply with single-cycle access to memory banks. We implemented a Meta State Table (MST) that tracks the search tree-building process while keeping a partitioned copy of the tree state matrix, eliminating performance-prohibitive pointer-based addressing. The algorithm was implemented on a Xilinx ALVEO U280 FPGA, resulting in a power-efficient, high-performance design. Hardware optimizations were applied to adapt the computational profile of the algorithm to better exploit both data and pipeline parallelism on the FPGA.

The proposed FPGA implementation achieves a significant improvement in the decoding time compared to CPU and GPU implementations while preserving BER performance. The FPGA design shows up to $9\times$ speedup compared to the CPU implementation, allowing what was previously prohibitive to be viable. The 15×15 MIMO configuration with 4-QAM modulation meets the real-time constraint of $10ms$ at an SNR of 12 dB on the optimized CPU implementation; on the other hand, the FPGA implementation decoded the signal in real-time at an SNR of 4 dB. Moreover, the FPGA design can scale to 20×20 MIMO installations while satisfying the real-time response at an SNR of 8 dB. The FPGA's power efficiency excelled by showing an average energy savings of $38.1\times$ compared to the CPU implementation. Furthermore, we achieved an average speedup of $57\times$ compared to the GPU's GEMM-BFS based approach by refactoring the algorithm and incorporating a more efficient search strategy.

The hardware optimizations introduced in this paper are not only to improve decoding time but also to optimize the FPGA resource utilization. This opens up promising research for future work to further parallelize the execution of the SD algorithm by partitioning the search tree over multiple Processing Entities (PEs). Additionally, the FPGA's flexible hardware architecture facilitates the use of low-precision number representations, which may further improve performance and reduce resource usage. Hence, another promising extension of this work is to explore the impact on BER performance and decoding time when using half-precision (FP16) and mixed-precision implementations.

REFERENCES

[1] 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.

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

[3] I. S. Association *et al.*, "Ieee std 802.11-2016, ieee standard for local and metropolitan area networks—part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications."

[4] 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.

[5] 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.

[6] A. Dabah, H. Ltaief, Z. Rezki, M.-A. Arfaoui, M.-S. Alouini, and D. Keyes, "Performance/complexity trade-offs of the sphere decoder algorithm for massive MIMO systems," *arXiv preprint arXiv:2002.09561*, 2020.

[7] 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.

[8] 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.

[9] L. G. Barbero and J. S. Thompson, "Fixing the complexity of the sphere decoder for mimo detection," *IEEE Transactions on Wireless communications*, vol. 7, no. 6, pp. 2131–2142, 2008.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] R. Gowaikar and B. Hassibi, "Statistical pruning for near-maximum likelihood decoding," *IEEE Transactions on Signal Processing*, vol. 55, no. 6, pp. 2661–2675, 2007.

[17] 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.

[18] C. Husmann, R. Tafazolli, and K. Nikitopoulos, "Antipodal detection and decoding for large multi-user mimo with reduced base-station antennas," in *IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018.

[19] C. Husmann, G. Georgis, K. Nikitopoulos, and K. Jamieson, "Flexcore: Massively parallel and flexible processing for large mimo access points," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 197–211.

[20] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing opencl kernels for high performance computing with fpgas," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis(SC)*, 2016.

[21] M. W. Hassan, A. E. Helal, P. M. Athanas, W.-C. Feng, and Y. Y. Hanafy, "Exploring fpga-specific optimizations for irregular opencl applications," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2018.

[22] Xilinx. Vitis BLAS library. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ibraries/vitis-blas.html

[23] 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

[24] 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, pp. 771–777.

[25] R. University. Wireless open-access research platform (warp). [Online]. Available: http://warpproject.org