Performance / Complexity Trade-offs of the Sphere Decoder Algorithm for Massive MIMO Systems

A. Dabah[1], H. Ltaief[1], Z. Rezki[2], M.-A. Arfaoui[3], M.-S. Alouini[1], and D. Keyes[1]

[1]Computer, Electrical and Mathematical Science and Engineering,
King Abdullah University of Science and Technology
{Adel.Dabah.1, Hatem.Ltaief, Slim.Alouini, David.Keyes}@kaust.edu.sa
; Adabah@cerist.dz

[2]University of Idaho, Moscow, ID USA,
zrezki@uidaho.edu

[3]Concordia University, Montreal, Canada,
m_arfaou@encs.concordia.ca

Massive Multiple-Input Multiple-Output (Massive MIMO) systems are seen by many researchers as a paramount technology toward next generation networks. This technology consists of hundreds of antennas that are capable of sending and receiving simultaneously a huge amount of data. One of the main challenges when using this technology is the necessity of an efficient decoding framework. The latter must guarantee both a real-time response complexity and a good Bit Error Rate (BER) performance. The Sphere Decoder (SD) algorithm represents one of the promising Maximum Likelihood (ML) decoding algorithms in terms of BER. However, it is inefficient for dealing with large MIMO systems due to its prohibitive complexity. To overcome this drawback, we propose to revisit the sequential SD algorithm and implement several variants that aim at finding appropriate trade-offs between complexity and performance. We conduct experiments to assess the critical impact of the SD components, i.e., the exploration strategies and the evaluation process, and to accelerate the search for the optimal combination of the transmitted vector. Then, we propose an efficient high-level parallel SD scheme based on the master/worker paradigm, which permits multiple SD instances to simultaneously explore the search tree, while mitigating the overheads from load imbalance. The results of our parallel SD implementation outperform the state-of-the-art by more than 5× using similar MIMO configuration systems, and show a super-linear speedup on multicore platforms. Moreover, this paper presents a new hybrid implementation, which enables to simulate MIMO configurations at an even larger scale. It combines the strengths of SD and K-best algorithms, i.e., by maintaining the low BER of SD, while further reducing the complexity using the K-best way of pruning search space. The hybrid approach extends our parallel SD implementation: the master contains the SD search tree and the workers use the K-best algorithm to accelerate its exploration. The resulting hybrid approach enhances the diversification gain, and therefore, lowers the overall complexity of our parallel SD algorithm. Our synergistic hybrid SD-K-best approach permits to scale up large MIMO configurations up to 100×100 using modulations with dense constellations, without sacrificing the BER and complexity. To our knowledge, this is the first time near-optimal results are reported on such a MIMO dimension in the literature.

Index Terms—**Massive MIMO Systems, Sphere Decoder Algorithm, K-Best Algorithm, Parallel Multicore CPU Implementations.**

I. Introduction

Multiple-Input Multiple-Output (MIMO) technology represents a generalization of Single-Input Single-Output (SISO) technology that increases the capacity of a radio link by sending multiple data streams at the same time [1]–[4]. Due to their obvious advantages, MIMO systems have already been incorporated into many wireless communication network protocols [5], [6] such as IEEE 802.11n (Wi-Fi), IEEE 802.11ac (Wi-Fi), etc. Massive MIMO is a new emerging technology that aims to amplify all the benefits of a traditional MIMO by further scaling the number of antennas up to several hundreds. In 5G era and beyond (6G), and the advent of the Internet of Things (IoT) [1], massive MIMO systems are viewed by many researchers and industrials as one of the key technologies to sustain a high spectral efficiency in communication networks [7], [8]. The challenge in these networks resides in the huge number of connected devices, exchanging enormous quantities of data (voice, video, etc.) under a real-time response constraint. In addition to this challenge, increasing the number of antennas raises several problems, especially in terms of energy efficiency and complexity caused

[1]a technology by which various devices, domestic, and industrial appliances are equipped with an IP address and incorporated into a wireless network of "things".

by the signal decoding procedure. Indeed, when scaling up the number of antennas, decoding a message becomes one of the most time-consuming operations. In order to maintain a real-time response, researchers generally use linear decoders, which are characterized by low complexity with a real-time response, but poor performance in terms of Bit Error Rate (BER). In order to achieve near-optimal signal decoding, researchers rely on the Maximum Likelihood (ML) and Sphere Decoder (SD) algorithms [9]–[11]. The ML decoder performs a brute force exploration of all possible combinations in the search space of the transmitted vector. Its complexity increases exponentially with the number of antennas making it impossible, in practice, to deploy for massive MIMO systems. The SD algorithm is another near-optimal decoder derived from the ML that reduces the size of its search space, thus, lowering its complexity. Indeed, the SD algorithm compares only the received vector with those solutions inside a sphere of a given radius. The radius of the sphere impacts the complexity and the BER of the overall MIMO system: the smaller the radius, the lower the search space (i.e., the complexity), but at the cost of possibly missing the actual sent vector if the radius is too small. Tuning the radius is paramount not only to identify the actual sent vector, but also to execute the corresponding procedure under real-time constraints. Nevertheless, it turns out that for massive MIMO systems, the resulting search space may still be too large to operate on and may engender high complexity.

In this paper, we address the massive MIMO scaling challenge by adapting the SD algorithm to achieve both decent BER performance and acceptable time complexity. To this aim, our contributions are centered around the following three levels.

The first level of our contributions focuses on revisiting the SD sequential algorithm and optimizing the time complexity of its main components. The SD algorithm operates on a search tree, where leaf nodes represent all possible combinations of the transmitted vector. Its goal is to find the combination (leaf node) with the minimum distance from the received signal. Two essential aspects of this algorithm must be taken into consideration: (1) how to efficiently explore the search tree, i.e., which node to select first, and (2) how to optimize the evaluation process, i.e., the process of computing the distance of each search tree node from the received signal. We assess the critical impact of different exploration strategies on the complexity of the SD algorithm, namely: Breadth-First Strategy (BFS), Depth-First Strategy (DFS), and Best-First Strategy (Best-FS). We further reduce the complexity of the SD algorithm by reformulating the evaluation process in terms of matrix algebra to increase the arithmetic intensity. We additionally introduce an incremental evaluation in order to avoid redundant computations. The idea here is to compute the evaluation of a current node by reusing the evaluations of its previous parent node. By choosing Best-FS as the optimal exploration strategy and performing these two aforementioned optimization techniques, we significantly reduce the complexity of the sequential SD algorithm, while maintaining an optimal error rate performance.

The second level of our contributions focuses on accelerating the sequential SD algorithm by using parallel multicore CPU architectures. Our proposed parallel implementation relies on the master/worker paradigm. It exploits the fact that each path in the SD search tree can be explored in an embarrassingly parallel fashion. Indeed, the search tree may be recursively divided into several smaller search trees where each one is explored by an instance of SD. Several instances of the SD algorithm may simultaneously explore the search tree, i.e., one instance of the SD algorithm operating as a master process and the others as workers. This parallel version aims to diversify the search process, which may rapidly reduce the radius and thus, the complexity. This method, called *diversification gain*, allows to avoid the exploration of a huge number of branches explored in the serial version. However, due to the irregular workload on each path, the parallel implementation may run into a load balancing problem, which may affect its parallel scalability. To overcome this drawback, we propose an efficient dynamic load balancing strategy, which adjusts the workload per thread at runtime. Our proposed parallel approach using our load balancing strategy reports more than 5× speedup compared to a recent work from Nikitopoulos et al. [12] on a similar 10×10 16-QAM MIMO configuration. It also achieves up to 60× speedup compared to our serial SD version using a 16-QAM modulation on a two-socket 10-core Intel Ivy Bridge shared-memory platform (i.e., 20 cores total). This represents a super-linear speedup, which has been possible thanks to the diversification gain. It turns out that even when using parallelism, the complexity of our SD algorithm may still be very high to deal with larger MIMO systems and constellation sizes.

To further reduce the complexity, the third level of our work involves a trade-off between the complexity and the performance, via a new hybrid implementation combining the strengths of our parallel SD and the K-best algorithms. The main idea of this new implementation, code-named SD-K-best, is to accelerate the exploration of the SD search tree stored on the master process by using several workers with the low-complexity K-best algorithm. This approximate method permits to explore rapidly and partially the subtree sent by the master, which reduces effectively the complexity. The selected nodes (i.e., branches/paths) are chosen according to their partial distance from the received signal. Thus, they are more likely to contain good solutions and may eventually ensure a satisfactory BER. Our synergistic SD-K-best implementation integrates all benefits of the parallel SD algorithm (i.e., diversification gain, Best-FS, and sphere radius) to increase the chances of encountering good combinations of the transmitted signal, while reducing effectively the complexity using the parallel SD implementation associated with the K-best algorithmic strengths. The obtained results of our SD-K-best implementation show an overall low complexity and good performance in terms of BER, as compared to the reference K-best algorithm. Indeed, for a 16×16 MIMO system using 64-QAM modulation, our SD-K-best approach reaches acceptable error rate at a 20 dB Signal-to-Noise Ratio (SNR) and real-time requirement (i.e., 10 ms) starting from 28 dB. Last but not least, our SD-K-best approach shows a strong scalability potential by reporting acceptable complexity and good error rate performance for a $100{\times}100$ MIMO system using 64-QAM modulation. To our knowledge, such a record has never been achieved previously in the literature.

The remainder of the paper is organized as follows. Section II summarizes literature on solving massive MIMO systems. Section III describes the system model, recalls the components of the conventional SD algorithm, and details its exploration strategies. Section IV presents our proposed SD algorithm with a new exploration strategy and introduces new optimization techniques. Details of our serial and parallel multicore implementations are shown in Section V. Section VI highlights our new hybrid SD-K-best implementation, which is necessary to tackle massive MIMO systems. Results and discussions about the trade-off between complexity and performance are given in Section VII. Finally, Section VIII concludes this paper and summarizes our future perspectives.

II. Related Work

In recent years, there has been a significant body of work dealing with massive MIMO systems due to their important role in next generation networks. Indeed, many surveys have been proposed in the literature highlighting open challenges and recent advances in this area [13]–[16]. With the recent proposed graphene-based plasmonic nano-antenna arrays that can accommodate hundreds of antenna elements in a few millimeters; Massive MIMO will continue to play an important role in 6G communication networks using Terahertz-Band [7], [17], [18].

Signal decoding in massive MIMO represents the most challenging and critical task since the performance of the whole system, in terms of Bit Error Rate (BER), depends on it. Signal decoding consists in estimating the transmitted vector by taking into account the received vector, which is subject to noise. Two kinds of decoders exist in the literature: linear and non-linear (near-optimal) decoders. Due to the complexity of non-linear decoders, there are only few works in the literature that actually deal with massive MIMO systems. Most of these works explore partially the Maximum Likelihood (ML) search tree using the Sphere Decoder (SD) algorithm and/or leverage high performance computing architectures (e.g., GPUs) to accelerate the search process and to increase the throughput.

In [19], Roger et al. propose a parallel fixed complexity SD for MIMO systems with bit-interleaved coded modulation. Their parallel approach exploits multicore processors to compute the preprocessing phase of the algorithm, and the massively GPU hardware resources to process simultaneously the detection phase for all N sub-carriers in the system.

In [20], Jozsa et al. propose a GPU-based SD algorithm for multichannel (i.e., sub-carriers) MIMO systems. Their approach performs multiple detections simultaneously on the GPU, which increases the throughput. Moreover, a second level of parallelism introduced within each detection relies on the GPU thread block to accelerate the exploration process of the SD algorithm.

In [21], Wu et al. propose an improved version of their initial parallel decoder [22] to increase the throughput of a flexible N-way MIMO detector using GPU-based computations. This problem consists in dividing the available bandwidth into multiple sub-carriers. Each sub-carrier corresponds to an independent MIMO detection problem. Therefore, the receiver needs to perform multiple MIMO detection procedures. The authors' idea is to use multiple GPU blocks to execute multiple MIMO detection algorithms simultaneously. To support multiple detections on the GPU, the authors use a soft-output MIMO detection, which engenders a low memory footprint. The results show a good throughput, outperforming the results presented in [19].

The main problems with the above approaches are twofold. The scalability is a serious bottleneck for large numbers of antennas due the limited amount of GPU memory in presence of multi-carriers. Moreover, the high latency increases the complexity due to the slow PCIe interconnect, when performing data movement between CPU host and GPU device.

In [23], Chen and Leib propose a GPU-based Fixed Complexity Sphere Decoder (FCSD) for large-scale MIMO uplink systems. The authors reported a speedup around $7\times$ for large MIMO systems and constellation sizes compared to their CPU implementation. However, the time complexity of their approach is significant even for small numbers of antennas.

In [24], Arfaoui et al. propose a GPU-based SD algorithm in which a Breadth-First exploration Strategy (BFS) is used to increase the GPU resource occupancy. However, increasing the GPU hardware utilization using BFS increases the complexity due to the limited impact of pruning process, especially in low Signal-to-Noise Ratio (SNR). Our optimized sequential SD implementation herein achieves up to 255-fold speedup on a similar 25×25 MIMO system with BPSK constellations.

In [25], Christopher et al. propose a parallel flexible decoder for large MIMO systems using GPU and FPGA architectures. Their algorithm contains two phases. A first preprocessing phase chooses parts of the SD search tree to explore, and a second phase maps each of the chosen parts of the SD tree to a single processing element (GPU or FPGA). The results are presented for a 12×12 MIMO system using a 64-QAM modulation.

In [12], the authors propose the design and implementation of a parallel multi-search SD approach for large MIMO search tree using multicore CPU and Very-Large-Scale Integration (VLSI) architectures. After the preprocessing phase in which they obtain a processing order of the tree branches, the authors split the search tree into several sub-trees. Each sub-tree is then mapped on a processing element and explored using a depth-first strategy. However, the authors do not take into consideration the load balancing problem, which may arise in modulations with dense constellations. They also do not update the sphere radius at runtime, which may negatively affect the time complexity of their parallel implementation. The authors report optimal results for a 10×10 MIMO system using 16-QAM modulation and approximate results for a 16×16 MIMO system using 64-QAM modulation.

Most of the existing works report experimental results for rather small MIMO configuration systems and do not report or satisfy the real-time response constraint. In addition, they rely on GPUs to accelerate the partial or complete exploration of SD search-trees. While GPUs are throughput-oriented devices, the resulting size of the SD search space still remains prohibitive to maintain a decent time complexity.

We decide herein to revisit the fundamentals of the popular serial SD algorithm, which stands as a proxy for all non-linear decoders. We reduce its time complexity by relying on a new Best-First Strategy (Best-FS) for efficient exploration, a matrix algebra reformulation for increasing arithmetic intensity, and an incremental evaluation process for cutting down the number of operations. These optimizations are performed for all SNR regions allowing to reduce the time compexity, while maintaining optimal BER performance. We then extend the sequential implementation by exploiting the inherent parallelism of the SD algorithm. We take advantage of the diversification gain to avoid the exploration of a huge number of branches explored in the serial version. We employ a dynamic load balancing scheduler that minimizes idleness, communications, and synchronization overhead. Finally, in order to break the symbolic barrier of hundreds of antennas for the first time, we deploy a new hybrid approximate approach that blends the aforementioned strengths of our SD implementation with the ones from the K-best algorithm. This new SD-K-best CPU-based implementation achieves performance and complexity metrics at unprecedented levels from the literature, even with GPU hardware accelerators.

III. System Model and Conventional SD Algorithm

A. System Model

In this paper, we consider a baseband MIMO system consisting of M transmit antennas and N receive antennas, as depicted in Figure 1. The transmitter sends M data streams simultaneously to a receiver using multiple antennas via a flat-fading channel. This system is

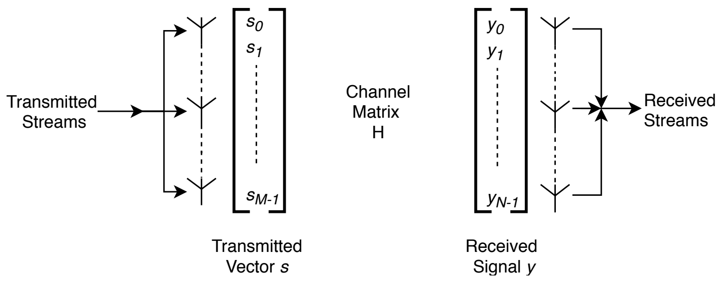


Figure 1: Example of a MIMO system where the vector $\boldsymbol{s}$ is transmitted by M transmitter antennas via a channel matrix $\boldsymbol{H}$. The received vector $\boldsymbol{y}$ is a collection of N receiver antennas' observations.

described by the input-output relation in the following Equation 1:

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{s} + \boldsymbol{n}, \tag{1}$$

where the vector $\boldsymbol{y} = [y_1, ..., y_N]^T$ represents the received signal. $\boldsymbol{H}$ is an $N \times M$ channel matrix, where each element $\boldsymbol{h_{ij}}$ is a complex Gaussian random variable that models the fading gain between the j-th transmitter and i-th receiver. The vector $\boldsymbol{s} = [s_1, ..., s_M]$ represents the transmitted vector, where $\boldsymbol{s_i}$ belongs to a finite alphabet set denoted by Ω. Finally, $\boldsymbol{n} = [n_1, ..., n_N]^T$ represents the additive white Gaussian noise with zero mean and covariance $\boldsymbol{I}_N$, where $\boldsymbol{I}_N$ designates the identity matrix of size N. For convenience, let us consider $\boldsymbol{S}$ as the set of all possible combinations of the transmitted vector $\boldsymbol{s}$. The possible number of combinations corresponds to the complexity of the MIMO system and it is calculated as follows: $|\boldsymbol{S}| = |\Omega|^M$.

There are two options to decode the received signal. Either we use linear decoders characterized by low complexity and poor performance in terms of BER, or we use non-linear (optimal) decoders characterized by good BER quality but high complexity.

Linear decoders multiply and map the received signal using a matrix denoted by $\boldsymbol{H_{inv}}$ ($M \times N$), obtained from the channel matrix $\boldsymbol{H}$. The most commonly used linear decoders in the literature define $\boldsymbol{H_{inv}}$ as follows:

- Maximum Ratio Combining (MRC), where the H_{inv} is equal to:
$$\boldsymbol{H_{inv}} = \boldsymbol{H^H}.$$
- Zero Forcing (ZF), where the H_{inv} in case of $M \leq N$ is equal to:
$$\boldsymbol{H_{inv}} = (\boldsymbol{H^H} \cdot \boldsymbol{H})^{-1} \cdot \boldsymbol{H^H}.$$
- Minimum Mean Square Error (MMSE), where the H_{inv} matrix used by this decoder is equal to:
$$\boldsymbol{H_{inv}} = (\boldsymbol{H^H} \cdot \boldsymbol{H} + \frac{1}{SNR} \cdot \boldsymbol{I_m}) \cdot \boldsymbol{H^H},$$
with the Signal-to-Noise Ratio $SNR = P$, where P is the average transmit power, since we normalize

the noise covariance to identity, without loss of generality.

As for non-linear decoders, the Maximum Likelihood (ML) is the *de facto* decoder, exhibiting high complexity. It calculates *a posteriori* probability for each possible transmitted vector $\boldsymbol{s} \in \boldsymbol{S}$. In other words, the algorithm performs a brute-force exploration of the entire search space, as shown in the following Equation 2:

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

The ML decoder chooses the vector s that minimizes the distance between the received vector y and the assumed transmitted vector $\boldsymbol{Hs}$. In perfect conditions, i.e., in absence of noise, this minimum distance is equal to zero, which indicates that the transmitted vector is exactly the received one, up to a channel multiplication. For more details about the ML decoder, the reader may refer to [26]. Another example of non-linear decoders is the Sphere Decoder (SD) algorithm [10], [27]. This latter mimics the ML decoder, but limits the search for the candidate vector to a smaller space than ML, reducing enormously the complexity. The SD algorithm consists in exploring solutions inside a sphere of radius r set initially by the user, as shown in the following Equation 3:

$$||\boldsymbol{y} - \boldsymbol{Hs}||^2 < r^2, \quad where\ \boldsymbol{s} \in \boldsymbol{S}. \tag{3}$$

The radius may then be updated subsequently during the search process at runtime to further prune the search space and reduce the complexity. In the following section, we describe the baseline SD algorithm and its components in more details.

B. Conventional Sphere Decoder Algorithm

The SD algorithm operates on a search tree that models all possible combinations of the transmitted vector. This algorithm aims to find the best path in terms of distance from the received signal, while ignoring non promising branches. Equation 3 can be translated in solving the integer least-square problem. It starts with a preprocessing operation by performing a $\boldsymbol{QR}$ decomposition of the channel matrix $\boldsymbol{H}$ as $\boldsymbol{H} = \boldsymbol{QR}$, where $\boldsymbol{Q} \in \mathbb{C}^{N \times N}$ is an orthogonal matrix and $\boldsymbol{R} \in \mathbb{C}^{N \times M}$ is an upper triangular matrix. This preprocessing step permits to expose the matrix structures of $\boldsymbol{Q}$ and $\boldsymbol{R}$, which will be eventually used to simplify the computations. Indeed, by using the orthogonality of $\boldsymbol{Q}$ and considering only the $M \times M$ upper part of $\boldsymbol{R}$, the problem defined in the Equation 3 can be transformed into another equivalent problem as follows:

$$||\boldsymbol{y} - \boldsymbol{Hs}||^2 = ||\boldsymbol{y} - \boldsymbol{QRs}||^2$$
$$= ||\boldsymbol{Q}(\boldsymbol{Q}^{\boldsymbol{H}}\boldsymbol{y} - \boldsymbol{Rs})||^2$$
$$= ||\boldsymbol{Q}^{\boldsymbol{H}}\boldsymbol{y} - \boldsymbol{Rs}||^2$$
$$= ||\bar{\boldsymbol{y}} - \boldsymbol{Rs}||^2,\ where\ \bar{\boldsymbol{y}} = \boldsymbol{Q}^{\boldsymbol{H}}\boldsymbol{y}$$
$$= || \begin{bmatrix} \bar{y}_0 \\ \bar{y}_1 \\ \cdot \\ \cdot \\ \cdot \\ \bar{y}_{M-1} \end{bmatrix} - \begin{bmatrix} r_{00} & r_{01} & \cdots & & r_{0M-1} \\ 0 & r_{11} & \cdots & & r_{1M-1} \\ \cdot & \cdot & \cdots & & \cdot \\ \cdot & \cdot & \cdots & & \cdot \\ \cdot & \cdot & \cdots & & \cdot \\ 0 & 0 & \ldots & 0 & r_{M-1M-1} \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ \cdot \\ \cdot \\ \cdot \\ s_{M-1} \end{bmatrix} ||^2.$$

Therefore, finding the supposed transmitted vector ($\hat{s}$) in Equation 1 is equivalent to solving the following minimization problem:

$$\min \sum_{k=1}^{M} g_k(s_{M-1}, ..., s_{M-k}), \tag{4}$$

where $g_k(s_{M-1}, ..., s_{M-k}) = ||\bar{y}_{M-k} - \sum_{i=M-k}^{M-1}(r_{(M-k),i} s_i)||^2$. This latter formulation of the problem allows us to model all possible combinations of the transmitted vector (i.e., search space) as a search tree with M layers. To find the path with the minimum distance from the received signal, the SD algorithm is decomposed into three components: branching, evaluation, and pruning. Figure 2 shows an example of the SD search tree and highlights the SD components on a MIMO system with three transit antennas using the Binary Phase-Shift Keying (BPSK) modulation.

Algorithm 1: The Sphere Decoder (SD) Algorithm.

```
Data: Received signal $y$
      Constellation order $\Omega$
      Channel estimation $H$
      Noise variance estimation $\sigma^2$
      Radius $r$
Result: Decoded vector $\hat{s}$
initialization;
List < -- root;
while List != ∅ do
    $P$ = select_node (List);
    List = List - {$P$};
    Generate successors $P_i$ of $P$ / $i = \{1, ..., |\Omega|\}$;
    for each $P_i$ do
        if $E(P_i) < r$ then
            if $P_i$ is a leaf node (complete solution) then
                r= $E(P_i)$;
                (The evaluation of sub-problem $P_i$.)
                $\hat{s} = F_{P_i}$ ;
            else
                List = List ∪ $P_i$;
            end
        else
            prune the branch;
        end
    end
end
```

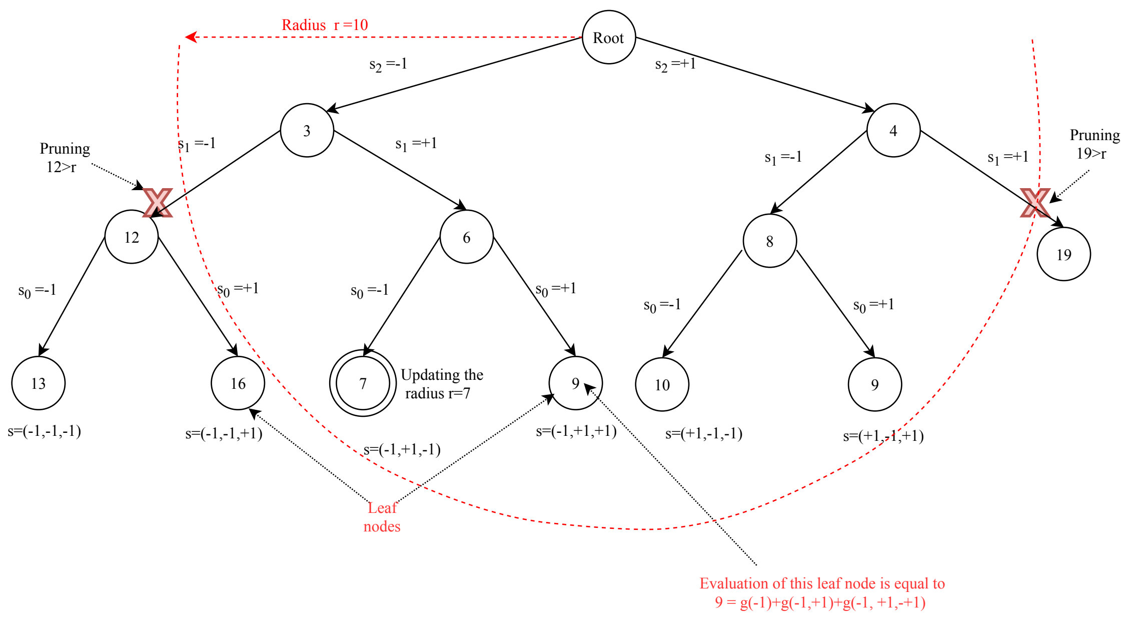


Figure 2: This figure represents an example of the SD search tree for a MIMO system with three transmit antennas. At each level, one symbol is fixed, starting with the last one. The partial evaluation of each node (sub-problem) is stored inside the circles. The pruning process uses the evaluation of the node and the sphere radius r to avoid unpromising branches.

In the following, we describe formally each of the SD components, as described in Algorithm 1.

1) *Branching*

the branching component for a MIMO system with M transmit antennas is performed over the symbols of a transmitted vector. This process creates a search tree with M levels, so that each level corresponds to one symbol. Thereby, the last level of the search tree contains all possible combinations of the transmitted vector. Each search tree node is characterized by a set of fixed symbols denoted by F. In this way, there is no fixed symbols in the root node ($F_{root} = \emptyset$). The branching component is essentially a recursive process that divides the search space related to a search tree node P over several successors (or sub-problems) $P_i, i = 1, ..., |\Omega|$. Each subsequent successor is eventually handled in the same way until a complete solution is found, i.e., until the number of symbols in the solution is equal to M. The number of immediate successors depends on the size of the alphabet. For example in Figure 2, the size of the constellation in the BPSK modulation is two (-1 and +1). In this case, we have two immediate successors (P_1,P_2) of node P, where P_1 is characterized by the set F_{P1}=$F_p \cup \{-1\}$ and P_2 is characterized by the set F_{P2}=$F_p \cup \{+1\}$. At each level of the search tree, we perform branching over one symbol. Since matrix $\boldsymbol{R}$ from the $\boldsymbol{QR}$ decomposition is upper triangular, each level l of the search tree corresponds to a symbol $\boldsymbol{s_{M-l}}$. For example, level 1 corresponds to symbol $\boldsymbol{s_{M-1}}$ and level M corresponds to symbol $\boldsymbol{s_0}$. Thus, we begin by fixing $\boldsymbol{s_{M-1}}$, then symbol $\boldsymbol{s_{M-2}}$, and so on, until we reach the leaf nodes where symbol $\boldsymbol{s_0}$ is fixed.

2) *Evaluation*

the evaluation component represents the process of computing the Partial Distance (PD) of each search tree node from the received signal. After the branching process, the evaluation, denoted by E in Algorithm 1, is calculated for each successor using Equation 4. More precisely, the evaluation of a search tree node P characterized by L fixed symbols ($F_p = \{\boldsymbol{s_{M-1}}, ..., \boldsymbol{s_{M-L}}\}$) is defined as $E(P) = \sum_{k=1}^{L} g_k(\boldsymbol{s_{M-1}}, ..., \boldsymbol{s_{M-k}})$. This comes down to calculate:

$$E(P) = \| \begin{bmatrix} \bar{\boldsymbol{y}}_{\boldsymbol{M-L}} \\ . \\ . \\ \bar{\boldsymbol{y}}_{\boldsymbol{M-1}} \end{bmatrix} - \begin{bmatrix} 0 & . & .... & r_{M-L,M-2} & r_{M-L,M-1} \\ . & . & .... & . & . \\ . & . & .... & . & . \\ 0 & 0 & ... & 0 & r_{M-1,M-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{s_{M-L}} \\ . \\ . \\ \boldsymbol{s_{M-1}} \end{bmatrix} \|^2.$$

This means that we use only the last L elements in vector $\bar{\boldsymbol{y}}$ and the last L lines in matrix $\boldsymbol{R}$ to compute the evaluation of a node P with L fixed symbols where $L = |F_p|$.

3) *Sphere Radius and Pruning*

the sphere radius defines the region of the search space in which an intelligent enumeration can be performed. The radius represents an important parameter for determining the complexity of the SD algorithm, since a large value of the radius induces a high complexity. The ideal value for the radius should as small as possible, but as long as the corresponding region still includes the ML solution. The sphere radius, denoted by r, imposes an upper limit for the expression $||\bar{\boldsymbol{y}} - \boldsymbol{Rs}||^{\boldsymbol{2}}$, which leads to reject any partial combination of the transmitted vector $\boldsymbol{s}$ with a partial evaluation greater than r. The sphere radius used in this paper is equal to $r^2 = N \cdot M \cdot 10^{\frac{-SNR}{10}}$

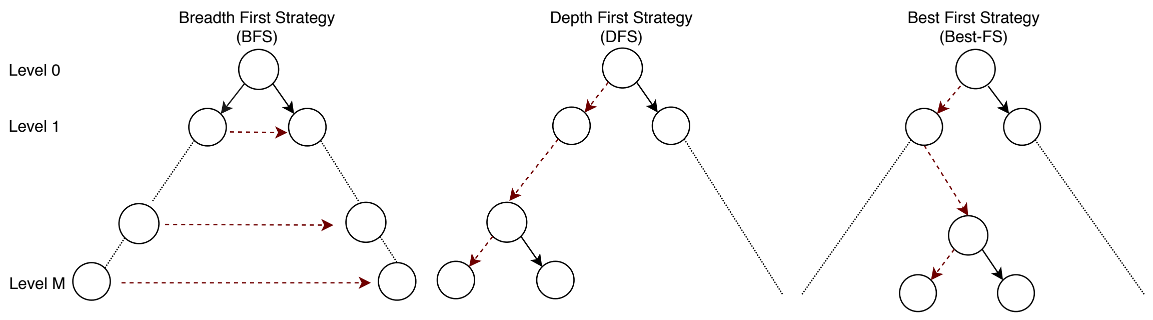


Figure 3: Exploration strategies used by the SD algorithm.

and is the same used in [24]. Tuning and studying the impact of r on complexity and performance is critical but beyond the scope of this paper.

The pruning process consists in detecting and eliminating the unpromising branches in the search tree by using both the sphere radius and the evaluation of nodes. As seen in Equation 4, the evaluation increases each time we fix a new symbol in the transmitted vector. This means that a node P with a partial evaluation $E(p) >= r^2$ can not lead to a complete solution that improves the best one already found. In this specific case, the node is eliminated. In order to ensure an efficient pruning process during the search, we replace the value of the radius each time a new better solution (leaf node) $\boldsymbol{s} \in \boldsymbol{S}$ is explored, i.e., $r^2 = E(\boldsymbol{s})$. Updating the value of the radius during the pruning phase is very important for the subsequent explorations of the search tree. This feature may prevent exploring a huge number of branches that are outside of the sphere radius.

C. Exploration Strategies

To give an idea about the magnitude of the search space, the number of combinations (leaf nodes) for a MIMO system with the BPSK modulation and fifty transmit antennas is 1.1258999 e^{+15}. Exploring all these possibilities under real-time constraints is prohibitively expensive. The exploration strategies for the SD algorithm define the way the search tree is explored and traversed, as illustrated in line 4 of Algorithm 1. The performance of each exploration strategy depends on the MIMO configurations and the underlying hardware architectures. For this reason, we investigate in this paper the impact of several exploration strategies on the SD complexity. The SD search tree is stored using a list structure and is partially explored at each iteration. Figure 3 shows the two typical ways of exploring a tree: Breadth-First and Depth-First.

1) Breadth-First Traversal Strategy

the Breadth-First Strategy (BFS) explores the search tree level by level, which means that all nodes of a given level must be explored before moving toward the lower levels. In practice, implementing the BFS consists to apply First-In First-Out (FIFO) strategy on the data list that contains the tree, i.e., selecting always the rightmost node in the list. The BFS is particularly very suitable for parallel implementation since all nodes of a given level can be treated independently. This enables to efficiently exploit the available computing resources. However, its major drawback is the high memory footprint during the search process. This makes its application very limited in practice, especially for massive MIMO systems where the number of possible solutions may be tremendous. The second major drawback of the BFS strategy is the fact that the sphere radius remains the same throughout the search process, since this strategy reaches the leaf nodes only at the last level. The static sphere radius cannot be updated at runtime and engenders a poor pruning process, which induces a very high complexity even for small MIMO systems.

2) Depth-First Traversal Strategy

the Depth-First Strategy (DFS) is a recursive process based on a backtracking technique. Unlike the BFS, the DFS aims to reach leaf nodes as quickly as possible by exploring down the current path. Once it reaches the leaves, DFS may explore backward to retrieve new nodes and carry on again along the new path until attaining the bottom of the tree. This process is pursued until all nodes are explored. In practice, implementing the DFS consists to apply a Last-In First-Out (LIFO) strategy to the data list that contains the tree. In other words, the DFS always select the leftmost node in the list, which is the most recently added to the list after the branching process. The interesting fact that should be highlighted for DFS in general is the limited memory usage. This feature makes it very suitable in practice for challenging problems such as decoding messages in massive MIMO systems. The other interesting fact is the possibility of updating the value of the sphere radius dynamically due to the huge number of entirely explored solutions. Although the complexity may be improved for sequential implementations, the DFS does not expose parallelism compared to BFS. Therefore, it may not be suitable in presence of high number of computing resources needed to operate large MIMO configurations.

IV. Leveraging The Sphere Decoder Algorithm Toward Massive MIMO

In this section, we present an initial set of optimization techniques to improve the exploration and the evaluation phases in our SD algorithm.

A. Best-First Strategy

In order to improve the search tree exploration, we introduce the Best-First Strategy (Best-FS). This strategy is very similar to the DFS since both are meant to explore leaf nodes first. However, the Best-FS targets a better quality of leaf nodes (in terms of distance from the received signal) as compared to the DFS exploration model. After the branching process, Best-FS chooses first the node with the best evaluation in order to complete its exploration. The only difference against the DFS model is that the nodes generated after the branching process are sorted according to their partial distance before being inserted into the list. Since the number of nodes generated after the branching is limited, the overhead time of the sorting process is insignificant. The exploration based on Best-FS is theoretically more suited for SD implementation since it targets better quality leaf-nodes. Therefore, this approach proactively reduces the sphere radius throughout during the SD process, which decreases the number of explored nodes and thus, the memory footprint and the arithmetic complexity.

After optimizing the exploration phase, we aim to further reduce the SD complexity by optimizing its evaluation phase. This latter represents the most time-consuming part of the SD algorithm, since it is calculated for each search tree node. To achieve this goal, we consider two aspects: reducing the number of evaluation steps and avoiding redundancy in the evaluation process.

B. Grouping Evaluation Steps

The idea here is to reduce the number of intermediate evaluation points for each path in the search tree. For a MIMO system with M transmit antennas, we generally perform M evaluation points to reach the leaf nodes, which may be overwhelming when scaling up massive MIMO systems. Reducing the number of evaluation points can be achieved by performing the branching process simultaneously over several symbols instead of one at a time. Indeed, performing the branching over J positions in the transmitted vector allows us to reduce the number of evaluation points from M to M/J. For instance, performing the branching over five symbols for a MIMO system with 100 transmit antennas will reduce the number of evaluation points for each search tree path from 100 to only 20. Beside shortening the overall processing time of the evaluation phase, this grouping technique allows to reach the leaf nodes more quickly which may result not only in reducing the latency overhead but also in pruning earlier a lot of branches.

However, we should keep in mind that the number of immediate successors will increase according to the number of fixed symbols in the branching process. Therefore, instead of creating $|\Omega|$ new successors, we create $|\Omega|^J$ new successors. The parameter J should be tuned accordingly to trade-off complexity and parallelism.

C. Incremental Evaluation

Since the search tree for massive MIMO may be huge, it is paramount to optimize the evaluation in order to achieve good performance. Our goal here is to further reduce the complexity of the evaluation step by avoiding redundant computations.

We recall that the evaluation of a search tree node P with L fixed symbols is equal to $E(P) = \sum_{k=1}^{L} g_k(\boldsymbol{s_{M-1}}, \ldots, \boldsymbol{s_{M-k}})$. We can see that the complexity of the evaluation increases significantly when moving toward leaf nodes. In order to avoid this increase in complexity and to have the same evaluation time for all search tree nodes, we take advantage of the incremental nature of the evaluation process for this problem. Indeed, the evaluation of successors P_i of the node P with L_i fixed symbols, where $L_i > L$, can be decomposed as follows:

$$E(P_i) = \sum_{k=1}^{L_i} g_k(\boldsymbol{s_{M-1}}, \ldots, \boldsymbol{s_{M-k}})$$
$$= \overbrace{\underbrace{\sum_{k=1}^{L} g_k(\boldsymbol{s_{M-1}}, \ldots, \boldsymbol{s_{M-k}})}_{\text{E(P)}} + \underbrace{\sum_{k=L+1}^{L_i} g_k(\boldsymbol{s_{M-1}}, \ldots, \boldsymbol{s_{M-k}})}_{\text{non-computed part}}}. \tag{5}$$

Therefore, we accumulate the calculations during the evaluation of previous nodes in order to use it later when evaluating the successors. In this way, the evaluation process for all search nodes consists to compute only the non-computed part of Equation 5.

V. Implementation Details of The Parallel Sphere Decoder Algorithm

This section provides details of the sequential and parallel Sphere Decoder (SD) implementations based on the Best-FS combined with the grouping and incremental evaluation steps.

A. Serial Implementation of the SD Algorithm

In the following, we describe our idea of optimizing the implementation of the evaluation phase using the well-known Basic Linear Algebra Subprograms (BLAS).

As mentioned earlier, performing the branching over J symbols at a time allows us to reduce the number of evaluation steps. However, it also increases the number of successors from $|\Omega|$ to $|\Omega|^J$. Each successor is characterized by a vector of fixed symbols called here v. To evaluate all successors at once, we propose the following: We begin by regrouping all successors' vectors in one matrix named V, with $|v|$ lines (number of fixed symbols) and $|\Omega|^J$ columns (number of successors). After that, we create a matrix $\boldsymbol{R'}$ by considering only the last $|v|$ lines and the last $|v|$ columns of the matrix $\boldsymbol{R}$. Thereafter, we use the well-known BLAS library [28] to compute

$B = Y^{\star} - RV$, where $Y^{\star}$ represents the last $|v|$ elements of $\bar{y}$ duplicated $|\Omega|^{J}$ times. Finally, we use the matrix B to deduce the evaluation of each successor by computing the norm over the columns of matrix B. In this way, we have been able to optimize the implementation of this algorithm.

However, searching for the optimal combination of the transmitted vector is a time consuming operation due to the large scale of the SD search tree. In this section, we study the impact and possible gain of exploiting the processing elements of a single workstation. Indeed, most of today's machines are parallel from a hardware perspective; offering a decent computing power, which is not exploited in most cases. For this reason, and to evaluate the possible gain that can be achieved using small number of computing resources, two parallel SD approaches are proposed.

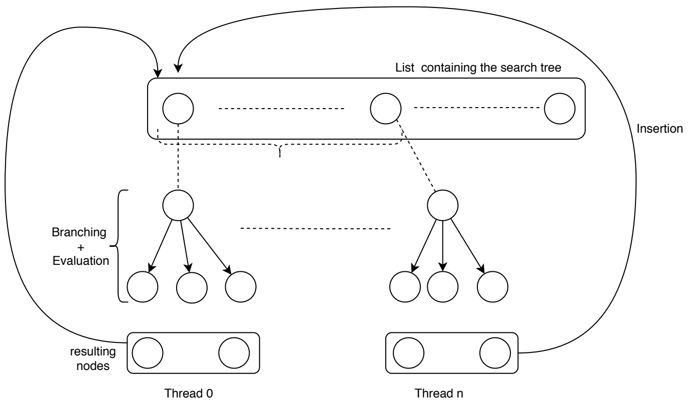


Figure 4: Low-level SD parallelization.

B. Low-level Parallel SD Approach

Our first attempt to accelerate the SD algorithm is by accelerating its exploration process. As depicted in Figure 4, this first approach aims to accelerate the sequential process of exploring one search tree stored in a list. At each iteration, the serial SD algorithm takes a search tree node from the list and performs the branching operation, which creates a set of successor nodes. After-that, the SD calculates the partial distance (evaluation) for all resulting nodes before adding them to the list. The idea of this parallel approach is to perform the branching overs several nodes at a time. Therefore, at each iteration of the SD algorithm, we create a set of threads to perform the same sequential process (branching and evaluation) over several search tree nodes in a concurrent safe way. This process is repeated until the list becomes empty. Hence, the end of the parallel algorithm is reached. The number of threads and work-load for each one must be adapted to the number of processing elements available in the machine. Moreover, to ensure low memory utilization, this approach uses the Best-FS exploration model.

The interesting fact about this approach is the fair work-load distribution between parallel threads, which prevents the idleness especially for this kind of problems. The down side of this approach is the scalability issue that may occur when increasing the number of parallel threads, due to the concurrent access to the same data-structure. To avoid this problem, a second parallel SD approach is proposed.

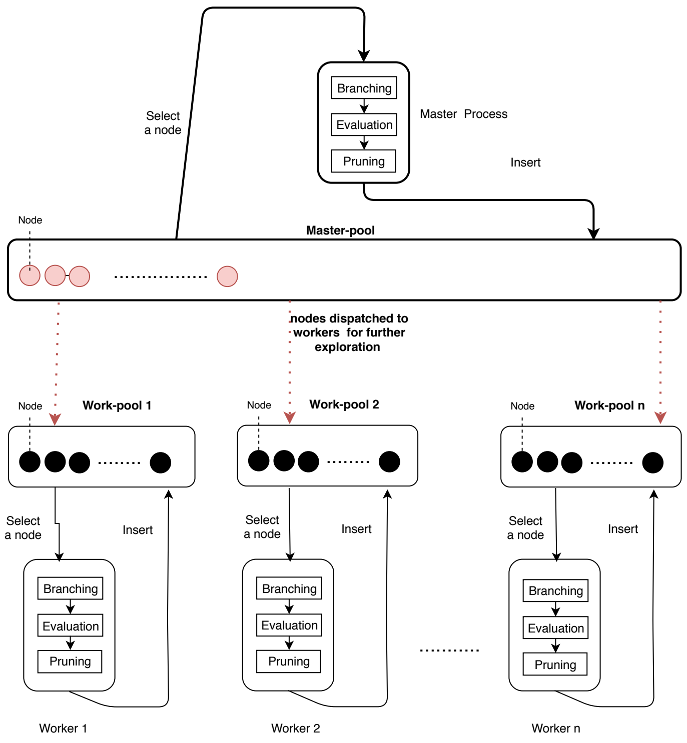


Figure 5: High-level SD parallelization.

C. High-level Parallel SD Approach

As depicted in Figure 5, our second parallel scheme can be seen as a high level parallelization in which several instances of the SD algorithm explore simultaneously the search space. Indeed, this scheme exploits the fact that the global tree which models all possible combination of the transmitted vector can be divided into several smaller subtrees where each can be explored independently from the others. The only shared information between the SD instances is the value of the sphere radius; which is updated each time a new better solution is explored by parallel threads. The proposed parallel scheme, which exploits the multi-core CPU processors, is based on the *Master/Worker* paradigm. According to this paradigm, we have one instance of the SD algorithm playing the role of the master process and other SD instances as workers. The master divides the search tree into several sub trees, which are meant to be explored by workers. We define a work-pool as a set of active nodes generated by the SD algorithm during the search process. Two kind of work-pool can be identified: a master-pool owned by the master process, and several local work-pools owned by the different workers. Initially, all workers are blocked, waiting for nodes to explore. The master creates the root node and begins the exploration of the search tree which generates a set of nodes in the master-pool. When the number of nodes in the master-pool is greater than the number of workers, the master wakes up blocked workers by sending to each one of them a node (subtree). After that, each worker launches its own SD instance to explore the sub-tree related to the received node. In order to efficiently reduce the sphere radius, all parallel SD instances (threads) explore their subsequent subtree according to the Best-FS model. The master periodically checks on the state of workers and wakes-up any blocked one (worker with an empty work-pool). Each time the master-pool is empty, the master checks the state of all workers. If all of them are blocked, the master sends an end signal to all parallel threads.

Load-balancing problem: Due to the prohibitive complexity of SD, and the irregular work-load in the SD sub-trees, a load-balancing strategy must be used. This latter has an objective to increase the efficiency of our high-level parallel SD approach by avoiding the idleness of workers. In our case, the idleness of workers appears only in the case where the master work-pool is empty. Our idea to avoid the idleness of workers is to perform a workload redistribution over all blocked workers whenever the master work-pool is empty. In this case, the master locates the worker which has the highest number of unexplored nodes. Then, it distributes them over blocked workers and move the most of remaining nodes to its own work-pool. In this way, we have been able to ensure a fair work-load distribution during the decoding process.

The parallelization allows to speedup enormously the exploration process and reduce the SD complexity. However, the complexity of the latter is still very high to deal with massive MIMO systems under real-time constraint. For this reason, we propose in the following a new approximate algorithm that perform a trade-off between the complexity and the performance in terms of error rate.

Similarly to the DFS, the Best-FS may be less suitable for parallelization as compared to the BFS exploration model. The number of nodes handled at each iteration is limited, leading to a low hardware occupancy.

VI. Our Hybrid Sphere Decoder / K-Best Algorithm

The main idea of our proposed approximate approach is to achieve an acceptable BER in real-time complexity, i.e., losing a bit in performance (as compared to the SD algorithm), but wining in terms of complexity. The challenge is to find the appropriate balance to achieve both near ML performance and real-time response.

One of the best reference algorithms performing a trade-off between the complexity and the performance is the K-best algorithm [29]. Similarly to SD, the K-best algorithm [29] operates on a search tree that models all possible combinations of the transmitted vector (see Figure 6). It explores the search tree level by level according to the BFS model. However, the algorithm keeps only the best K nodes in terms of evaluation for further exploration, and the remaining nodes from the level are systematically removed. This process is repeated for each level until reaching the last one where leaf nodes (solutions) exist. Among these solutions, the algorithm returns the best one in terms of distance. Since the search tree of this algorithm contains k nodes in each level, the total number of explored nodes by this algorithm is equal to $(M-1) \times K$, where M refers to the number of transmit antennas. Thereby, this algorithm has a fixed complexity irrespective to SNR.

Moreover, the number of kept nodes (K) should be carefully considered since it impacts the complexity of the algorithm. On one hand, a large value of parameter K allows the algorithm to achieve a near SD performance in terms of BER. However, the algorithm complexity increases significantly and can even exceed the SD complexity. In addition to that, a large value of K induces a significant sorting overhead, making the complexity of K-best far from real-time response. On the other hand, a small value for parameter K reduces the complexity, however, the algorithm loses in performance in terms of BER. Moreover, the performance of the algorithm, in terms of BER, drops significantly for dense constellations. To overcome all these drawbacks, we propose a hybrid parallel algorithm, named SD-K-best that takes all the benefits of SD and K-best algorithms. Our hybrid approach also aims to reduce the complexity of our high-level parallel SD version, while taking benefit from the Best-FS exploration, the sphere radius, and the diversification gain.

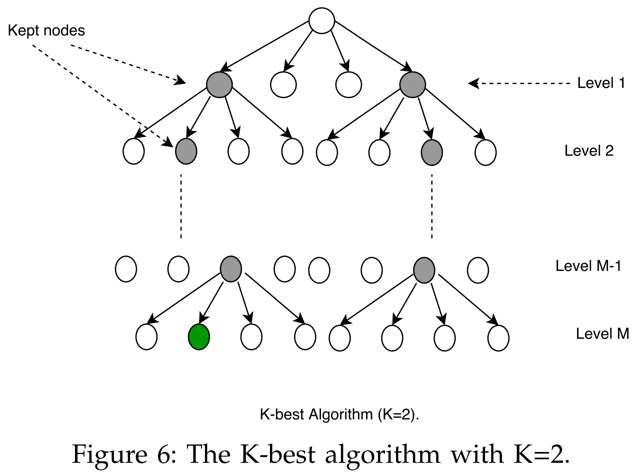


Figure 6: The K-best algorithm with K=2.

Indeed, our hybrid approach is based on our high-level parallel scheme, which means that a Master/Worker paradigm is also used in this approximate approach. The mater process executes a SD instance which builds the SD search tree in the master work-pool. To accelerate the exploration process of this tree, we use several workers with a low-complexity K-best algorithm. In other words, the master performs the pre-processing phase and generates the root of the SD search tree. After-that, it explores the search tree by a SD algorithm according to the Best-FS exploration mode which allows to explore first the most promising combinations. To accelerate the exploration process, the master sends nodes, from the head of the master work-pool (right-most nodes), for workers to complete their exploration. Each worker explores the received node by the mean of a K-best algorithm. Therefore, after the branching a worker keeps only the best K successors in terms of partial distance and ignore the rest which reduces the complexity. In-addition to the best K nodes selected by each worker, additional nodes may also be selected if their distance is very close to the distance of selected nodes. Moreover, since the value of parameter K used by each worker is small, there is an insignificant sorting overhead which allows workers to rapidly reach leaf-nodes and improve the radius. Improving the radius allows to reduce even more the complexity of workers. i.e., workers keep only the best successors which are inside the radius. This allows to target better quality combinations. In the case where the kept nodes by a worker are outside the radius, the corresponding worker ends its exploration and requests a new node from the master. The fact that the search tree is built in parallel allows to take benefit from the diversification gain which may allow to explore good combination and thus reducing more efficiently the radius and avoiding the explorations of huge number of branches. Finally, the ends of this hybrid approach is reached when the master work-pool is empty. The whole number of explored nodes by our hybrid SD-K-best is much bigger than the number of explored nodes by the K-best algorithm which leads to improve the BER performance. However, since this approach takes benefit of parallel architectures, the complexity of our hybrid SD-K-best can be less than the complexity of K-best algorithm, since the average number of explored nodes by a processing element is lower. In this way, we have been able to achieve both low complexity and good BER performance as demonstrated in the experiment results section.

VII. Experimental Results

In this section, we report the results of our proposed approaches. This section is organized in three parts. The first part shows the impact of different exploration strategies and optimization techniques on the SD complexity. The second part of this section reports the impact of using parallel architectures to accelerate the exploration process of the SD algorithm. Finally, the third part presents the results, in terms of BER performance and time complexity, of our proposed approximate algorithms for large MIMO systems.

We perform our experiments using a two-socket 10-core Intel Ivy Bridge CPU running at 2.8 GHz with 256 GB of main memory. Hyper-threading is enabled on the system in order to maximize resource occupancy. For all the experiments, we consider the case of a perfect channel-state information. This means that the channel matrix is known only at the receiver.

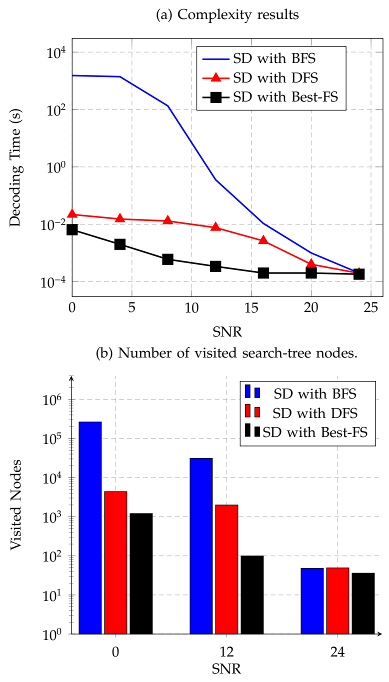


Figure 7: Impact of exploration strategies on SD performance for the 18 × 18 MIMO system using BPSK modulation.

A. Results of Optimizing the SD components

In the following, we report the results of using different exploration strategies and optimization techniques on the SD complexity.

Figure 7 reports the complexity and the number of visited tree nodes by our SD algorithm using different exploration strategies for a 18 × 18 MIMO system with BPSK modulation. The first result from Figure 7 is the positive impact of depth exploration strategies (DFS and Best-FS) on the SD algorithm complexity, as compared to the BFS exploration, especially in low SNR region. Indeed, the DFS is faster than BFS in low SNR region. This huge difference in complexity is the result of reducing the number of explored nodes, as reflected in Figure 7 (b). Indeed, the goal of the DFS is to reach leaf nodes quickly, which allows to explore a large number of combinations (leaf nodes). Thus, reducing the sphere radius and avoiding the exploration of non-promising branches, which reduces the complexity. This is not the case of the BFS. This latter explores the search tree level by level until reaching the last one where solutions exist. Therefore, the sphere radius remains the same during the decoding process. In-addition, the majority of nodes explored by the BFS belongs to lower levels of the search tree where the evaluation process has higher complexity as compared to the early levels of the tree. To summarize, the BFS explores huge number of nodes with

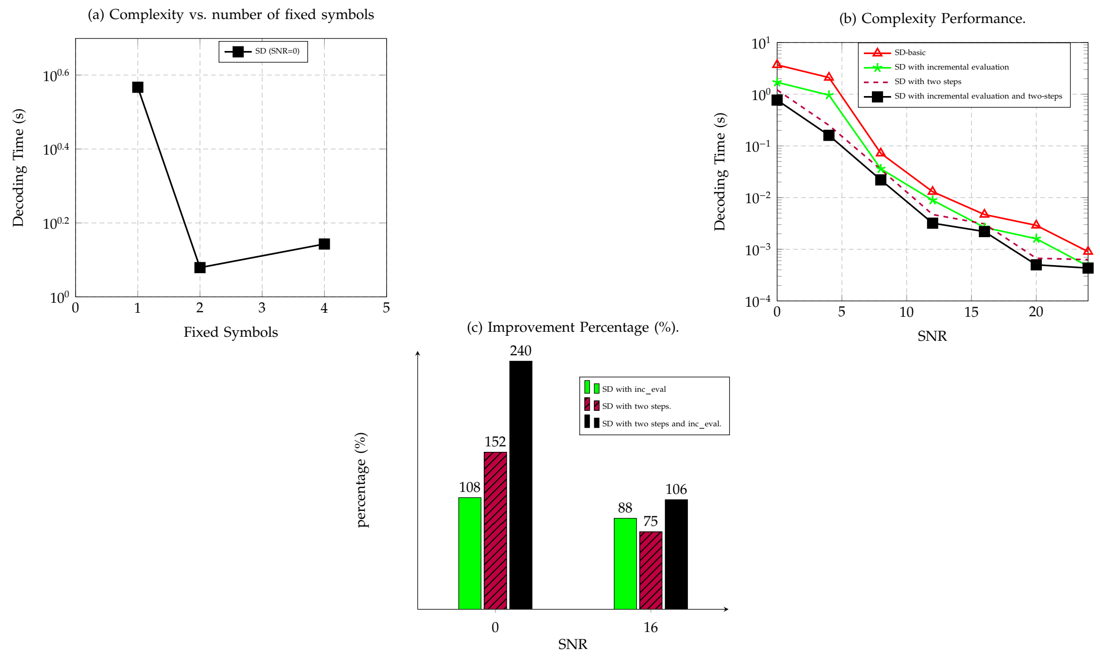


Figure 8: Impact of our optimization techniques on the SD performance for a 52×52 MIMO system using BPSK modulation.

high evaluation complexity as compared to the DFS. The latter avoids non-promising branches at early levels of the search tree. According to the same figures, the Best-FS performs even better as compared to the complexity of using the BFS and DFS strategies. Indeed, the Best-FS guides the search process toward better quality leaf nodes, which reduces the sphere radius more quickly and efficiently. Therefore, reducing further the number of explored nodes as compared to the DFS. This explains why the complexity of SD using the Best-FS is better than the complexity of the SD using the DFS. Figure 7 (a) also shows that the difference in complexity between the different exploration strategies decreases when increasing the SNR until reaching the same complexity for SNR equals to 24. This behavior is related to the estimation of the initial sphere radius. Indeed, the higher the SNR (lower noise) the better the estimation of the initial sphere radius. In fact, when reaching 24 dB, the initial sphere radius becomes smaller which induces an efficient pruning process for the BFS and equal number of explored nodes as we can see in Figure 7 (b). Moreover, Figure 7 (a) also shows that using the SD algorithm with Best-FS allowed us to reach real-time response ($10^{-2}s$) from 0 dB SNR, while the SD algorithm with DFS and BFS needs a SNR of 10 dB and 16 dB respectively to ensure a real-time decoding process. Thus, a difference of 16 dB in power consumption.

As a conclusion, the Best-FS is more suitable for serial implementation of the SD algorithm. For this reason, we will use it for all remaining experiments.

Figure 8 shows the impact of our optimization techniques on the complexity of SD algorithm for 52×52 MIMO system using BPSK modulation.

The first sub-figure (a) reports the complexity of performing the branching over several symbols at each iteration of the SD algorithm in low SNR region (0 dB). The subfigure shows that the best performance is reached when fixing two symbols at a time. After that, increasing the number of fixed symbols increases the complexity. This is mostly due to the increase in the number of resulting nodes that need to be evaluated; which induces an overhead in computation, especially when using a single CPU-core. For this reason, it is important that the number of fixed symbols fits well the targeted architecture and the number of its processing elements.

Similarly, sub-figures (b) and (c) report respectively the complexity over SNR and the improvement percentage of our optimizations in two SNR regions. The sub-figures show the positive impact of our incremental evaluation process on the complexity of the SD algorithm. The incremental evaluation process allowed us to increase the performance of the SD algorithm up to 108%, i.e., more than two times faster. Moreover, fixing two positions in the transmitted vector at a time, allowed an improvement in complexity up to 152%. Combining both optimizations allowed us to achieve even higher improvement to reach 240%. In addition to the improvement in complexity, we also have an improvement of four dB in power consumption as compared to the basic SD version. This improvement is the results of (1) grouping some evaluation steps and (2) avoiding redundancy in computing the evaluation for each search tree node.

Figure 9 shows the complexity of our approach as

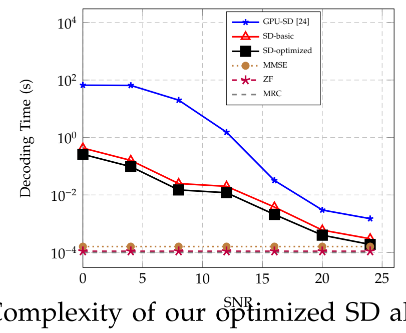


Figure 9: Complexity of our optimized SD algorithm as compared to our previous GPU-SD work in [24] for the 25×25 MIMO system.

compared to our previous work in [24] using a 25×25 MIMO system with BPSK modulation. The figure also shows the complexity results of the most used linear decoders in the literature: MMSE, ZF, and MRC. The 25×25 MIMO system represents the biggest that can be solved by the SD algorithm with BFS, due to the huge memory resources required by this strategy.

Figure 9 shows the importance of carefully optimizing the SD algorithm before moving toward its parallelization. Indeed, our optimized SD implementation is up to 255 times faster in the low SNR region and 15 times faster in the high SNR region as compared to the GPU-based SD in [24]. Figure 9 shows that, unlike linear decoders which have constant complexity, the complexity of the SD algorithm decreases when increasing the SNR. This behavior is closely related to the fact that the initial value of the sphere's radius, which is inversely proportional to the SNR. Therefore, higher the SNR, the smaller initial sphere radius, which reduces the search space, and thus, the complexity.

B. Results of our Parallel SD Approaches

In the following, we report the impact of using parallel architectures on the SD complexity. Two parallel approaches have been proposed. The first parallel approach (PL-SD) uses a set of threads to explore several nodes at a time; while the second parallel approach (PSD) uses simultaneously several instances of the SD algorithm to explore the search tree in parallel. Due to the unbalanced work-load for each instance, two version of the PSD are proposed depending on the nature of the used load-balancing strategy: PSD with static load-balancing (S-PSD) and PSD with dynamic load-balancing (D-PSD). In our PL-SD approach, creating and destroying threads at each iteration induces a considerable overhead time, which slows down considerably this parallel version. To overcome this problem, the workload for each parallel thread must be high enough to cover this overhead time. In our case, the workload for each thread is around twenty nodes.

Figure10 shows the impact of increasing the number of parallel threads on the time complexity of our parallel SD approaches. This time is measured for SNR equals

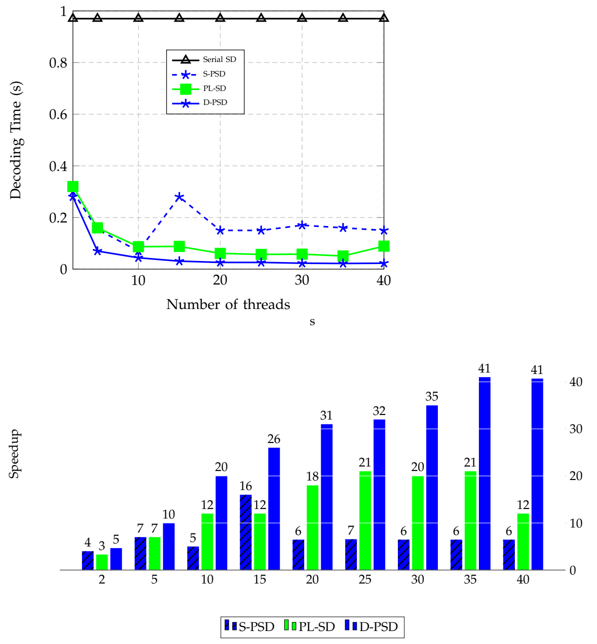


Figure 10: Impact of increasing the number of threads on the complexity of our parallel SD approaches for a 10×10 MIMO system using 16-QAM modulation.

to zero. Figure10 also shows the speedup obtained by our approaches for each number of threads. The first result from the figure is the positive impact of parallelism on reducing the complexity of the SD algorithm. As we can see in Figure 10, the curve of our low-level parallel approach has three phases. A first phase, between two and ten threads, characterized by a rapid decrease in complexity, when increasing the number of threads. This means that adding threads in this phase is beneficial and reduces the complexity. This is due to the positive impact of splitting the work-load over several processing elements (CPU-cores) and the low synchronization overhead. This overhead is related to the concurrent access to the same data structure that contains the search tree. This latter must be accessed by one thread at a time to have a valid execution result. After that, begins a second phase between ten and thirty-five threads where adding new threads has no impact of the decoding time. This can be explained by fact that the synchronization overhead neutralizes the gain of exploiting additional processing elements. After that, a third phase begins. This last is characterized by an increase in complexity when increasing the number of parallel threads. This behavior is related to the overhead of synchronization which increases when increasing the number of threads. Along with the sequential execution of threads by processing elements. As a result, an ideal number of threads for this low-level version will be between twenty and thirty for this parallel approach.

Figure 10 also shows the results of our high-level parallel approach with static and dynamic load-balancing strategies. This parallel version is based on splitting the search tree into several subtrees, where each one is explored independently by a parallel thread. The gain obtained by our high-level approach using a static load-balancing (S-PSD) strategy is limited due to the imbalanced work-load in each subtree. This results in a long execution time for few threads while others are ideal. By adding our dynamic load-balancing strategy (D-PSD), the performance of our high level parallel approach improved substantially. In fact, adding our proposed dynamic load-balancing strategy allowed us to reach a relative speedup of forty times faster using twenty CPU-cores and thirty-five parallel threads as compared to our optimized serial SD version. The curve of our high-level approach has two phases. A first phase characterized by a rapid decrease in complexity (increase in speedup), and a second phase where adding new threads does not change the complexity. The first phase, between two and twenty threads has a super-linear speedup. This speedup is the result of (1) low synchronization overhead since each thread explores its subtree independently from the others. Therefore, no concurrent access to the same data structure. (2) The fair work-load distribution among parallel threads due to our load-balancing strategy. This latter prevents the idleness of threads. (3) The diversification gain, that allows to reduce the explored search space as compared to the serial version. Indeed, dividing and exploring the search tree in parallel may result in a rapid improvement of the radius, which allows to avoid the exploration of several branches explored in the sequential version. This results in a super-linear speedup as in our case. Moreover, when the number of threads is greater than twenty which is the number of available processing elements (CPU-cores), a second phase begins. In this phase, the complexity of our high-level approach should be increasing due to the serial execution of threads by the processing elements. However, this is not the case. In fact, we still get performance improvement to reach 41x speedup when using forty threads. This behavior can simply be explained by the diversification gain that neutralizes the serial execution overhead of threads and improve the performance.

The scalability refers to the possibility of still improving the performance of our parallel approaches when using huge number of processing elements. According to Figure 10, our low-level approach does not scale well since all parallel threads operates on a single search tree. Therefore, inducing a synchronization overhead which effects the performance. This is not the case of our high-level approach which is embarrassingly parallel due to the low communication and synchronization costs between parallel threads, since each one operates on its own search tree. However, it needs a load balancing strategy to ensure a fair workload distribution among parallel threads.

Figure 11 shows the performance (complexity, speedup) of our parallel approaches when increasing the SNR for two MIMO systems (10×10 and 16×16) using

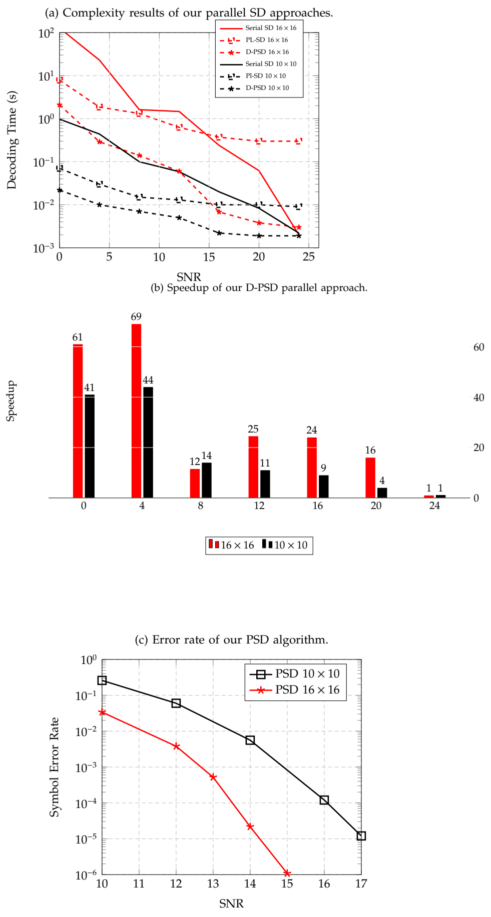
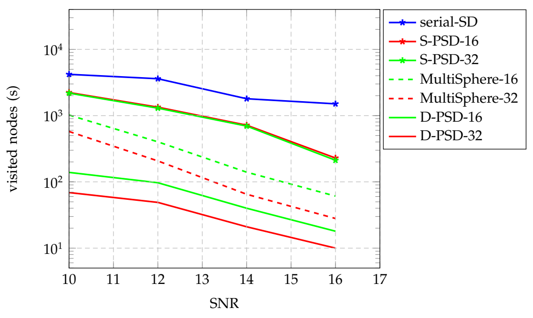


Figure 11: Complexity of parallel and sequential SD algorithms with 16-QAM modulation for 10×10 and 16×16 MIMO systems.

16-QAM modulation. Our both parallel approaches use thirty-five threads. Moreover, the speedup shown in this figure is obtained by our high-level parallel approach (D-PSD). The first observation is that the performance of our parallel approaches decreases when increasing the SNR. This is caused by the decrease in work-load for each parallel process when increasing the SNR since the sphere radius get smaller. Our high-level approach gives better performance, for these two MIMO configurations, as compared to our low level approach. This latter, does not perform well for high SNR and has higher complexity as compared to the sequential SD due to the synchronization overhead. The figure shows a good performance of our high-level parallelization scheme, especially for SNR between 0 and 8 db, where we have been able to reach a speedup around 69 times faster as compared to our optimized sequential SD algorithm. This speedup is obtained by using 35 threads and exploiting 20 processing elements (CPU-cores). This speedup is the result of: (1) reducing the synchronization overhead by choosing a good parallelization scheme, (2) exploiting efficiently the available computing resources by dividing fairly the work-load using load balancing strategy, and (3) taking

(a) Latency in terms of visited nodes.



(b) Complexity in terms evaluation (Partial Distance (PD)).

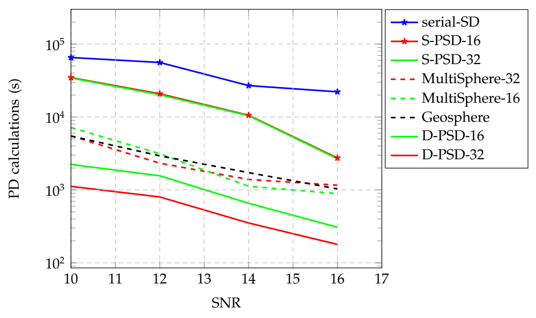


Figure 12: Our PSD's latency (a), and complexity (b) vs. our sequential SD and MultiSphere parallel approach for a 10×10 16-QAM MIMO.

advantage of the diversification gain which is not well studied in the literature. Moreover, Figure 11c shows the optimal Symbol Error Rate of our PSD for a 10×10 and 16×16 MIMO systems using 16-QAM modulation.

In the following, we compare our high-level parallel approaches against most recent works in the literature [12], [30]. Figure 12 shows a comparison between the latency and complexity of our PSD for two cases (16 and 32 threads) with the state-of-the-art parallel MultiSphere [12], Geosphere SDs [30], and our serial SD implementation for 16-QAM modulation using a 10×10 MIMO system. For fair comparison, the initial sphere radius is set to infinite. The number *visited nodes* refers to the average number of nodes (per thread) on which we performed a branching process, and the *PD calculations* refers to the average number of partial distance calculations, i.e., the average number of evaluated search tree nodes per thread. The PD calculations represents an important factor that highly influences the complexity of decoding approaches. For our S-PSD, these numbers are measured for the thread with the largest work-load, since the complexity of the parallel version depends depends on the complexity of this thread.

Figure 12 validates the results of our proposed parallel approach (PSD), especially when using dynamic load balancing strategy. For both 16 and 32 cases, the results of D-PSD outperform the results of parallel MultiSphere [12] and serial Geosphere [30] in terms of both visited nodes and PD calculations. Indeed, when increasing the number of threads from 16 to 32, our D-PSD consistently improves the performance in terms of both visited

(a) Complexity results of our SD-K-best approach.

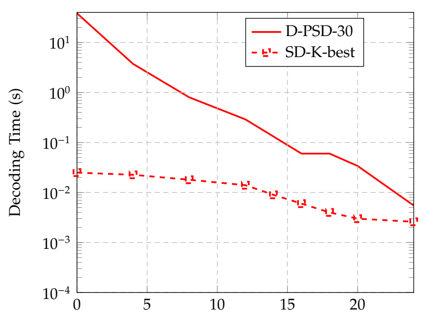


Figure 13: Time complexity of our SD-K-best algorithm vs. D-PSD algorithm for a 18×18 MIMO system using 16-QAM modulation.

nodes and PD calculations. This is not the case of the MultiSphere approach and our S-PSD and in terms of PD calculations due to the unbalanced workload between parallel threads. Moreover, our D-PSD reduces the complexity by a factor of 58x as compared to our serial SD and 5x as compared to the MultiSphere parallel approach.

C. *Results of Approximate Algorithms*

The parallelization allows to improve the performance of the SD algorithm especially, for large MIMO systems. However, the complexity remains very high to meet real-time response requirements for massive MIMO systems. To deal with such MIMO instances, the use of approximate algorithms is unavoidable. In the following, we report the preliminary results, as a proof of concept, of our proposed approximate algorithm SD-K-best and compare its performance against the well known K-best algorithm. The main idea of this hybrid algorithm is to accelerate the exploration of the SD search tree, stored in the master process, in a approximate way by using several low-complexity K-best algorithms (workers) running in parallel to meet real-time requirement.

Figure 13 shows a comparison between the performance, in terms of complexity, of our SD-K-best approach as compared to our Dynamic PSD approach for a 18×18 MIMO system with 16-QAM modulation. We can see from the figure that our SD-K-best performs better in terms of complexity as compared to our PSD since it explores partially the search tree. This makes it more applicable for various cases. Indeed, our SD-K-best approach is able to meet real time response from 13 dB, while our parallel PSD reaches this complexity at 23 dB; thus, 10 dB difference in power consumption.

Figures 14 and 15 show respectively the SER performance and complexity results for a 16×16 MIMO system using 64-QAM modulation. The figures compare the results of our SD-K-best algorithm with the results of K-best algorithm. Our SD-K-best algorithm uses twenty threads: one as a master process with a SD instance and nineteen as workers with K-best algorithm to accelerate the search process. Figure 14 shows the results

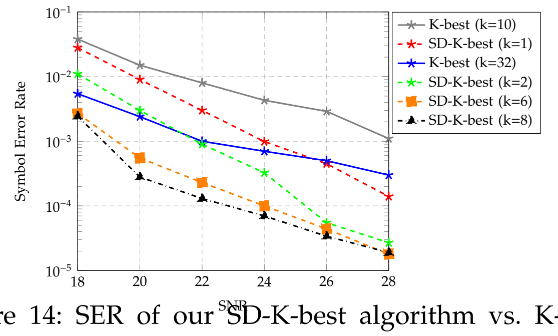


Figure 14: SER of our SD-K-best algorithm vs. K-best algorithm for a 16 × 16 MIMO system using 64-QAM modulation.

of our proposed SD-K-best with several configurations $k \in \{1, 2, 6, 8\}$. The figure also shows the results of the K-best algorithm with two configurations k=10 and k=32. The value of K refers to the number of kept nodes at each level by K-best algorithm, as well as workers in SD-K-best. The error rate for K-best algorithm with ten kept nodes does not reach the acceptable error rate (10^{-3}) even when reaching 26 dB. By increasing the value of K to 32, the performance of the latter is improved to meet this requirement in 22 dB. As compared to the K-best algorithm, our proposed SD-K-best algorithm performs better in terms of error rate and reach an acceptable rate even with small value of kept nodes. When increasing the number of kept nodes, the performance of our SD-K-best algorithm increases until reaching stagnation, i.e., increasing the number of kept nodes has minor impact on the SER. Moreover, our SD-K-best algorithm is able to reach acceptable error rate at a round 20 dB. Thus four dB improvement in power consumption as compared to the K-best algorithm with K equals to 32. The performance of our proposed approach in terms of error rate is explained by the large number of explored combinations as compared to a conventional K-best algorithm. Since each worker takes a subtree from the master and explores it according to the K-best algorithm, while taking benefit from the sphere radius to explore only promising branches which reduces the complexity.

Figure 15 (a) shows the average number of explored nodes per thread of our SD-K-best algorithm against the number of explored nodes by K-best algorithm for a 16 × 16 64-QAM MIMO system. In the same way, Figure 15 (b) shows the time complexity of our SD-K-best approach against K-best algorithm for the same MIMO system.

We can see from the Figure 15 that the complexity and the number of visited nodes by the K-best algorithm are fixed and stable across SNR. This is not the case of our SD-K-best algorithm since the average number of explored nodes per thread decreases when increasing the SNR. This behavior is closely related to the sphere radius which depends essentially on the SNR, i.e., higher the SNR the smaller the radius, thus reducing the number of explored nodes and the complexity. However, the whole

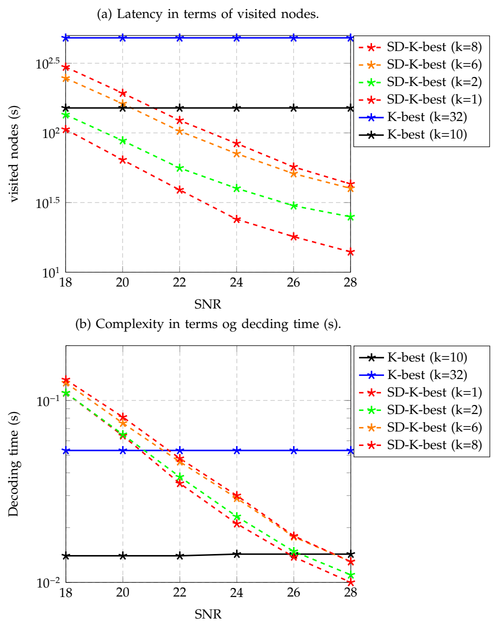


Figure 15: Our SD-K-best latency (a), and complexity (b) vs. K-best approximate approach for a 16 × 16 64-QAM MIMO.

number of explored nodes by all workers in SD-K-best algorithm is much bigger as compared to the K-best algorithm which explains the improvement in error rate. Moreover, Figure 15 (b) shows that the complexity of the SD-K-best algorithm is higher than the complexity of K-best algorithm in SNR between 18 and 22 dB region. This is due to the fact that the SD-K-best explores more search space than K-best algorithm, which explains why its complexity is higher. Furthermore, the high complexity is also explained by the complexity of the SD algorithm executed by the master process. However, this is not the case in the high SNR region (22 dB to 26 dB) where the SD-K-best algorithm has less complexity than K-best algorithm. This is the results of an efficient pruning process due to the small sphere radius in this region. In other words, the SD-K-best in the high SNR region explores less solutions, but they are of good quality as compared to K-best explored combinations. The challenge to deal with massive MIMO efficiently is based on finding the appropriate trade-off between the complexity and the performance in terms of error rate. To find it in our case, we combined strengths of both PSD and K-best algorithms to ensure low complexity and good BER at the same time. Unlike literature works, our proposed SD-K-best approach is able to reach both real time complexity and good error rate from SNR equals to 28 dB.

In the following, we scale-up the number of antennas to evaluate the ability of our hybrid SD-K-best algorithm to guarantee both low complexity and good error rate performance.

Figure 16 shows the obtained results in terms complexity and error rate performance, for a 100×100 MIMO

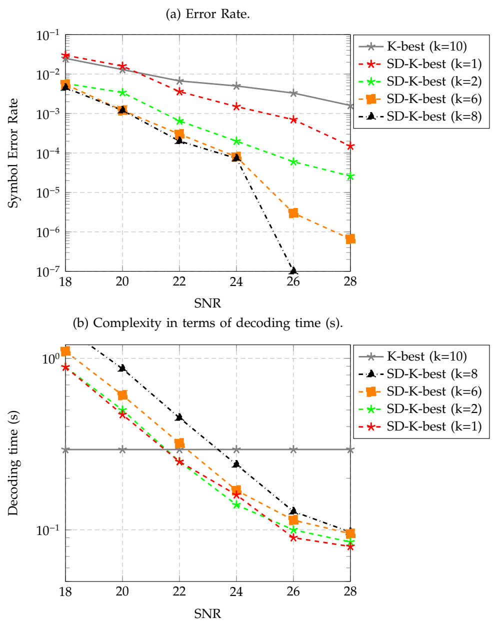


Figure 16: Error rate and complexity of our SD-K-best and K-best algorithms for a 100×100 MIMO system using 64-QAM modulation.

system with the 64-QAM modulation, i.e., an uncoded transmission rate of 400 bits per transmission. The first observation from the figure is the ability of approximate algorithms to deal with large MIMO systems. This is not the case of the SD algorithm due to its prohibit complexity. Figure 16 (a) shows the large improvement in terms of error rate performance of our SD-k-best algorithm with low number of kept nodes, as compared to K-best algorithm due to the diversification gain obtained from using parallelization. Moreover, starting from 22 dB, our algorithm has a low complexity than k-best algorithm due to the small radius in this region, which induces an efficient pruning process. Furthermore, increasing the number of kept-nodes for our SD-K-best algorithm have a good impact of the error rate, without increasing much the complexity as shown in Figure 16 (b). Indeed, the diversification gain improves more efficiently the radius which helps to reduce the overhead of increasing the number of kept-nodes.

In addition, the high SNR region in Figure 16 (b) shows that the curves begin to stabilize (floor) before even reaching the 10 ms threshold. In fact, when dealing with a larger number of antennas and constellation sizes, reaching a first leaf node can take more than 10 ms. This is due to the heavy computation needed at each level of the tree and the limited number of computing elements in CPU architecture. For this reason, it is critical to use highly efficient computing architectures with a large number of computing elements such as GPUs, which will be the subject of our future work.

VIII. Conclusion

We proposed in this paper the optimization of a well known optimal decoder with high complexity named, the Sphere Decoder (SD) algorithm. To achieve this goal, three levels have been considered. The first level consisted in optimizing the SD complements, especially the exploration strategies and the evaluation process since they have a huge impact on the complexity. Since the search tree for all the possible combination of the transmitted vector is huge, the second level aimed to speedup the search-tree exploration process by using parallel architectures. Finally, the third level consisted to use approximate algorithms that perform a trade-off between the complexity and the performance. The obtained results in each level confirmed our proposals and allowed us, not only to speedup the SD algorithm by a factor of 60× using a 16-QAM modulation, but also to deal with large MIMO systems with dense constellations, such as 100×100. To conclude this work, the challenge to deal with massive MIMO efficiently is based on finding the appropriate trade-off between the complexity and the performance in terms of error rate. To find it in our case, we combined the strengths of both parallel SD and K-best algorithms to ensure a low complexity and good error rate performance at the same time.

In the future, we plan to explore more parallel approximate algorithms to deal with a multi-user case situation using GPU architectures.

References

[1] E. Telatar, "Capacity of multi-antenna gaussian channels," *European transactions on telecommunications*, vol. 10, no. 6, pp. 585–595, 1999.

[2] G. J. Foschini, "Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas," *Bell labs technical journal*, vol. 1, no. 2, pp. 41–59, 1996.

[3] A. J. Paulraj and T. Kailath, "Increasing capacity in wireless broadcast systems using distributed transmission/directional reception (dtdr)," Sep. 6 1994, uS Patent 5,345,599.

[4] G. G. Raleigh and J. M. Cioffi, "Spatio-temporal coding for wireless communication," *IEEE Transactions on communications*, vol. 46, no. 3, pp. 357–366, 1998.

[5] E. Dahlman, S. Parkvall, J. Skold, and P. Beming, *3G evolution: HSPA and LTE for mobile broadband*. Academic press, 2010.

[6] S. McCann and A. Ashley, "Official ieee 802.11 working group project timelines," *November2011.[Online]. Available: http://www.ieee802.org/11/Reports/802.11_Timelines.htm*, 2014.

[7] S. Dang, O. Amin, B. Shihada, and M.-S. Alouini, "What should 6g be?" *Nature Electronics*, vol. 3, no. 1, pp. 20–29, 2020.

[8] B. M. Lee and H. Yang, "Massive MIMO for industrial internet of things in cyber-physical systems," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2641–2652, June 2018.

[9] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Mathematics of computation*, vol. 44, no. 170, pp. 463–471, 1985.

[10] E. Viterbo and J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Transactions on Information theory*, vol. 45, no. 5, pp. 1639–1642, 1999.

[11] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm i. expected complexity," *IEEE transactions on signal processing*, vol. 53, no. 8, pp. 2806–2818, 2005.

[12] 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*, 2018.

[13] X. Su, J. Zeng, L.-P. Rong, and Y.-J. Kuang, "Investigation on key technologies in large-scale MIMO," *Journal of Computer Science and Technology*, vol. 28, no. 3, pp. 412–419, 2013.

[14] L. Lu, G. Y. Li, A. L. Swindlehurst, A. Ashikhmin, and R. Zhang, "An overview of massive MIMO: Benefits and challenges," *IEEE journal of selected topics in signal processing*, vol. 8, no. 5, pp. 742–758, 2014.

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

[16] E. Björnson, E. G. Larsson, and T. L. Marzetta, "Massive MIMO: Ten myths and one critical question," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 114–123, 2016.

[17] H. Sarieddeen, M.-S. Alouini, and T. Y. Al-Naffouri, "Terahertz-band ultra-massive spatial modulation mimo," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 9, pp. 2040–2052, 2019.

[18] I. F. Akyildiz and J. M. Jornet, "Realizing ultra-massive mimo (1024× 1024) communication in the (0.06–10) terahertz band," *Nano Communication Networks*, vol. 8, pp. 46–54, 2016.

[19] S. Roger, C. Ramiro, A. Gonzalez, V. Almenar, and A. M. Vidal, "Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 8, pp. 3796–3800, 2012.

[20] C. M. Józsa, G. Kolumbán, A. M. Vidal, F.-J. Martínez-Zaldívar, and A. González, "New parallel sphere detector algorithm providing high-throughput for optimal MIMO detection," *Procedia Computer Science*, vol. 18, pp. 2432–2435, 2013.

[21] 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, no. 2, pp. 95–108, 2014.

[22] M. Wu, B. Yin, and J. R. Cavallaro, "Flexible n-way MIMO detector on GPU," in *2012 IEEE Workshop on Signal Processing Systems*. IEEE, 2012, pp. 318–323.

[23] T. Chen and H. Leib, "GPU acceleration for fixed complexity sphere decoder in large MIMO uplink systems," in *IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE), 2015*. IEEE, 2015, pp. 771–777.

[24] 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, international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050916308523

[25] C. Husmann, G. Georgis, K. Nikitopoulos, and K. Jamieson, "Flexcore: Massively parallel and flexible processing for large {MIMO} access points," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 197–211.

[26] M. K. Simon and M.-S. Alouini, *Digital Communication over Fading Channels (Wiley Series in Telecommunications and Signal Processing)*, 2nd ed. New York, NY, USA: Wiley-IEEE Press, 2004.

[27] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE transactions on information theory*, vol. 48, no. 8, pp. 2201–2214, 2002.

[28] BLAS., "Basic linear algebra subprograms. http://www.netlib.org/blas," 2013.

[29] K.-w. Wong, C.-y. Tsui, R.-K. Cheng, and W.-h. Mow, "A vlsi architecture of a k-best lattice decoding algorithm for MIMO channels," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, vol. 3. IEEE, 2002, pp. III–III.

[30] 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.