Massively Parallel Tree Search for High-Dimensional Sphere Decoders

Konstantinos Nikitopoulos, *Member, IEEE*, Georgios Georgis, *Member, IEEE*, Chathura Jayawardena, *Student Member, IEEE*, Daniil Chatzipanagiotis, and Rahim Tafazolli, *Senior Member, IEEE*

Abstract—The recent paradigm shift towards the transmission of large numbers of mutually interfering information streams, as in the case of aggressive spatial multiplexing, combined with requirements towards very low processing latency despite the frequency plateauing of traditional processors, initiates a need to revisit the fundamental maximum-likelihood (ML) and, consequently, the sphere-decoding (SD) detection problem. This work presents the design and VLSI architecture of MultiSphere; the first method to massively parallelize the tree search of large sphere decoders in a nearly-concurrent manner, without compromising their maximum-likelihood performance, and by keeping the overall processing complexity comparable to that of highly-optimized sequential sphere decoders. For a 10×10 MIMO spatially multiplexed system with 16-QAM modulation and 32 processing elements, our MultiSphere architecture can reduce latency by $29\times$ against well-known sequential SDs, approaching the processing latency of linear detection methods, without compromising ML optimality. In MIMO multicarrier systems targeting exact ML decoding, MultiSphere achieves processing latency and hardware efficiency that are orders of magnitude improved compared to approaches employing one SD per subcarrier. In addition, for 16×16 both "hard"- and "soft"-output MIMO systems, approximate MultiSphere versions are shown to achieve similar error rate performance with state-of-the art approximate SDs having akin parallelization properties, by using only one tenth of the processing elements, and to achieve up to approximately $9\times$ increased energy efficiency.

Index Terms—Sphere decoding, parallel processing, large multiple-input–multiple-output (MIMO), lattice search

1 Introduction

There is a general consensus that future mobile [2] and local area wireless communication systems [3], [4] shall be able to support very high peak user and network rates as well as very large numbers of connected devices, while keeping the latency requirements at very low levels. These needs have triggered a paradigm shift from *orthogonal* transmissions to systems where we intentionally transmit a large number of mutually interfering information streams, as in the case of multi-antenna (MIMO) deployments for aggressive spatial multiplexing. In this direction, and to keep detection complexity low, *large* and *massive* MIMO systems typically employ linear detection schemes, which however, can provide near-optimal performance only when the number of users is much smaller than the number of access-point or base-station antennas [5], [6]. Thus, typical large/massive MIMO deployments leave a large portion of the MIMO channel capacity unexploited, just for coping with the inefficiency of the linear detection approaches. Alternatively, maximum-likelihood (ML) detection schemes, allow to efficiently demultiplex as many mutually interfering information streams (e.g., spatially multiplexed users) as the number of the observed signals (e.g., base-station antennas) [5]. Still, even after translating the ML problem into a tree search, and solving it by means of sphere decoding [5], [7], [8] the corresponding processing and latency requirements increase exponentially with the number of mutually interfering information streams, substantially exceeding the processing capabilities of general purpose processors. These processing requirements, along with the-soon to be reached-plateau in the speed of microprocessors [9] prevent traditional systems from supporting large numbers of mutually interfering streams and, therefore, from scaling the achievable throughput gains and device connectivity.

At the same time, emerging system-on-chip architectures promise tens or even hundreds of cores per chip [10], something already feasible in graphics processing units (GPUs). In the presence of such multiple processing element (PE) architectures the complexity problem translates to the efficient utilization of available PEs or, equivalently, workload parallelization. Parallelizing the sphere decoder (SD) is a challenging task since its computational efficiency is determined by the ability to prune (i.e., exclude nodes from the tree search) large parts of the tree at an early stage of the tree search without compromising its algorithmic optimality. In order to achieve this, typical SD approaches providing the exact ML solution are of sequential nature. They start by finding a "good" candidate solution (i.e., one of relatively

• *The authors are with the 5G Innovation Centre, Institute for Communication Systems (ICS), University of Surrey, Guildford GU2 7XH, United Kingdom. E-mail: {k.nikitopoulos, g.georgis, c.jayawardena, r.tafazolli}@surrey.ac.uk, d.chatzipanagiotis@gmail.com.*

Manuscript received 30 May 2017; revised 28 June 2018; accepted 16 Sept. 2018. Date of publication 11 Oct. 2018; date of current version 11 Sept. 2019. (Corresponding author: Konstantinos Nikitopoulos.)
Recommended for acceptance by M. M Hayat.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2874002

small euclidean distance from the received signal) and they continue searching sequentially for "better" candidates without compromising the ML optimality while applying tree pruning strategies that become more aggressive any time a new candidate solution is found. Trivial parallelization approaches consisting of (nearly) independent parallel subprocesses, can result in less efficient tree pruning, and in turn, in highly increased processing requirements.

An "ideal" SD workload parallelization should be *scalable* and able to consistently reduce latency given additional processing power. It should be *complexity efficient* and not substantially increase the overall workload when increasing the number of PEs. For implementation purposes it should be *nearly "embarrassingly parallel"* and therefore minimize dependencies and communication overhead, which introduce latency and can moderate if not obliterate scalability and parallel efficiency [11]. In order to effectively allocate available processing power (PEs), an "ideal" parallelization method should also be *adjustable to the transmission conditions*. The methodology should be *generic* and *applicable to all kinds of SDs* including breadth-first and depth-first as well as exact (guaranteeing the ML solution) and approximate. Finally, it should be *transparent to the choice of the implementation platform*. In Many-Processor Systems on Chips (MPSoCs) for example, a PE can be a designated processor, in FPGA designs, a specifically allocated part of the chip and in GPU implementations the PE can be a separate thread.

Many SD implementations involve parallelism, but without meeting the above characteristics. For example, both depth-first [8], [12] and breadth-first [13] SDs perform several euclidean distance calculations in parallel, at each level of the SD tree, exploiting a limited degree of data parallelism. However, before the next data set can be processed in parallel, the necessary sorting operations introduce significant dependencies. This strategy is highly-dependent on the specific realization platform, inflexible, and cannot be efficiently employed to decrease latency in large SDs. Similarly, [14] proposes a low-dimensional, real-valued SD of limited degree of parallelism, accelerating the sequential case by only up to $2\times$ and only in low SNRs.

In GPU implementations, Khairy et al. [15] concurrently run multiple, low-dimensional (4×4) SDs without though parallelizing each SD. Wu et al. [16] and Jósza et al. [17] parallelize a low-dimensional MIMO detection process on GPUs. However, Wu et al. use a Trellis-decoder-like approximation of the SD which is not efficient for dense modulations and large MIMO systems, and Jósza et al. perform aggressive and nearly exhaustive parallel search of multiple subtrees without accounting for the overall complexity and by exhaustively trying different partitioning configurations. Hence, their approach is inappropriate for MPSoC or FPGA implementations and lacks theoretical reasoning.

Yang et al. [18], [19] propose a multi-core architecture for parallel high-dimensional SDs i.e., to the best of our knowledge the only other multi-core depth-first SD in the open literature. To parallelize the tree search, the SD tree in [19] is partitioned in subtrees consisting of only one node on the higher levels, and all possible nodes at the lower levels of the tree. The authors' SD partitioning starts first by allocating all the nodes of the higher level to subtrees, and each of the subtrees is then partitioned using the same principles provided that there are still available PEs. This partitioning strategy is very practical in terms of implementation, but cannot adjust to the transmission conditions [20], [21], [22]. Furthermore, to avoid visiting a node twice and control the overall complexity, Yang et al. employ an interconnection network to determine which of the nodes will be processed by which PE and to distribute the most promising solution from each of the subtrees. This reduces the flexibility of the approach, and therefore its efficiency when applied to platforms that require SIMD processing, as is the case of GPUs or implementations on individual processing blocks.

The fixed complexity SD (FSD) [23] sacrifices the ML optimality to acquire good parallelization properties. However, to efficiently parallelize such an SD, the available number of PEs should be a multiple of the order of the transmitted constellation. In addition, the way the FSD determines the tree paths to process in parallel, is pre-defined and cannot adjust to the transmission conditions. As shown in Section 3.2.4, our proposed approach can rectify these weaknesses. Koo et al. [24] propose a parallel version of the FSD, but with each concurrent set of tasks being followed by sequential operations. Consequently, their approach is bounded by the FSD's error-rate, and while the corresponding complexity can be smaller than FSD's this comes at the cost of random processing latency, in contrast to FSD. Moreover, [24] requires significant synchronization overhead per tree level, which together with the complex control/data flow of the processing element makes this approach unsuitable for anything besides a general purpose processor. This work proposes MultiSphere, the first SD of an "ideally" parallelized SD. As result, MultiSphere not only claims unexploited throughput in large MIMO uplink transmissions where traditional precoding approaches are infeasible, but it also facilitates efficient MIMO transmission without channel knowledge at the transmitter side (both in the uplink and the downlink), enabling high-throughput, and low-latency signal transmission as envisaged by future ultra-reliable, low latency mobile communication (uRLLC) systems. MultiSphere's unique characteristics originate from its ability to early examine the candidate vector solutions (i.e, SD tree paths) that are most likely to include the transmitted vector. We will hereafter refer to these most promising paths as *seeds*. The process of identifying the seeds can take place *a priori*, (i.e., before any information is received), and is based on the transmission characteristics (e.g., MIMO channel and signal-to-noise ratio). In this direction, in Section 3.1, we first introduce the concept of the *Tree of Promise* where the symbols constituting a candidate vector solution are described by their relative ordered distance to the received signal (e.g., kth closest symbol to the received signal), without though requiring the actual value of this received signal. Then, to each node in the *Tree of Promise* we assign a novel *Metric of Promise* (MoP) which approximates the actual probability of that node to be part of the transmitted vector (see Section 3.1.1). As described in Section 3.1.2, MultiSphere then employs a novel *tree partitioning method* which, based on the identified seeds and the number of the available PEs, splits the *Tree of Promise* (and equivalently the search space) into subtrees while preserving the ML optimality. In Section 3.1.3, MultiSphere employs a new method to *map* the actual transmitted (e.g., QAM) *symbols* to each *subtree* without performing any sorting operations, in order to preserve the complexity and energy efficiency of the approach. Then, when all subtrees are searched in a nearly concurrent manner, we propose a new *node traversal strategy and enumeration* in Section 3.2.1 that minimizes unnecessary euclidean distance calculations and applies to both sequential and the proposed parallel SD; in contrast to existing approaches ([5], [12], [25]) that only apply to sequential SDs.

In orthogonal multicarrier systems, like OFDM where there is no mutual interference between the subcarriers one can perform parallel detection by utilizing an exact, depth-first SD per subcarrier. However, as we discuss in Section 3.2.3, such a naive parallelization strategy is unable to efficiently reduce latency. On the other hand, as shown in Section 5.1, parallelizing each SD by means of MultiSphere and by sequentially processing each subcarrier, we can reduce latency by several orders of magnitude, for the same number of PEs and while preserving the ML optimality. Furthermore, in order to effectively exploit a large number of available PEs, Section 3.2.2 introduces a method that exploits the newly introduced MoPs to *adjust the number of allocated PEs* so that if their utilization is unable to significantly reduce latency, we can avoid PE redundancy and thus increased complexity. Then, based on this method, in Section 3.2.3 we propose a new *PE scheduling approach*, that allocates a variable number of PEs to different SD processes, and we show that such a scheme, is the first able to effectively reduce latency by increasing the number of PEs, while preserving ML optimality in a multicarrier multiantenna (MIMO) system.

MultiSphere has been evaluated in MIMO, spatially-multiplexed, multi-carrier systems using both mathematically modelled channels and actual channel traces collected in an indoor environment. Several versions of MultiSphere have been considered, ranging from exact “hard” to approximate “soft-output”. We show that compared to MIMO systems which exploit parallelism on a subcarrier level, MultiSphere can achieve two orders of magnitude reduction in processing latency at the same degree of parallelism while it can still exploit any amount of available PEs in order to further reduce this latency, even if that amount exceeds the number of subcarriers. Despite MultiSphere’s seemingly more complex structure (see Section 3.2.1), our novel VLSI architecture and effective design concepts (Section 4) show that MultiSphere’s VLSI processing throughput for exact detection with 32 processing elements for a 10×10, 16-QAM system is $29\times$ higher against efficient sequential detectors, thus validating MultiSphere’s efficiency. Similarly, for approximate soft-output detection our approach achieves almost an order of magnitude increased efficiency.

The rest of the paper is organized as follows. Section 2 introduces sphere decoding fundamentals. Section 3 presents MultiSphere’s design while Section 4 proposes an efficient VLSI architecture for both hard and soft-output MultiSphere. Section 5 extensively evaluates the proposed algorithmic design and VLSI architecture and Section 5.1 discusses MultiSphere’s extension to soft-output systems.

2 Sphere Decoding for MIMO Systems

For a spatially multiplexed MIMO system consisting of n_t transmit and n_r receive antennae the received signal vector is $\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{w}$, where $\mathbf{H}$ is the $n_r \times n_t$ MIMO channel matrix, $\mathbf{s}$ is the transmitted symbol vector whose elements belong to a constellation $\mathcal{O}$ of size $|\mathcal{O}|$ and $\mathbf{w}$ is the additive white Gaussian noise vector. By QR-decomposing the MIMO channel matrix as $\mathbf{H} = \mathbf{Q}\mathbf{R}$ the ML problem translates into finding $\hat{\mathbf{s}} = \arg\min_{\mathbf{s}\in\mathcal{O}^{n_t}} \|\widetilde{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2$, with R_{ij} being the elements of $\mathbf{R}$, and $\widetilde{\mathbf{y}} = \mathbf{Q}^{*}\mathbf{y}$ [5], [7], [8]. Since $\mathbf{R}$ is an upper triangular matrix, finding the ML solution can be transformed into a tree search of height n_t and branching factor $|\mathcal{O}|$. Each node at a level l can be characterized by its *partial symbol vector*$\mathbf{s}^{(l)} = [s_l,\ s_{l+1}, \ldots, s_{n_t}]$ which describes the path from the root to that node, as well as from its partial euclidean distance (PD) which can be calculated recursively as $d(\mathbf{s}^{(l)}) = d(\mathbf{s}^{(l+1)}) + c(\mathbf{s}^{(l)})$, where $c(\mathbf{s}^{(l)}) = \left|\widetilde{y}_l - \sum_{j=l}^{n_t} R_{lj}s_j\right|^2$ is the non-negative cost assigned to each branch. The ML problem translates then into finding the leaf-node with the minimum $d(\mathbf{s}^{(1)})$. In depth-first SDs with radius update and Schnorr-Euchner enumeration [5], [8], the initial squared radius $r^2 = \infty$. Any time a leaf node is reached for which $d(\mathbf{s}^{(1)}) < r^2$, r^2 is updated to $d(\mathbf{s}^{(1)})$. Upon meeting a node $\mathbf{s}^{(l)}$, if $d(\mathbf{s}^{(l)}) \geq r^2$ then this node, its children nodes and its siblings with all their descendants are excluded from the tree search (i.e., they are pruned). Following the Schnorr-Euchner tree-traversal [26] for node expansion, the nodes are visited in ascending order of their PDs. Since depth-first SDs with radius update and Schnorr-Euchner enumeration have been shown to be very efficient in practice [5], [8] and capable of delivering the ML solution, this is the structure that we will adopt for all our parallel SDs.

3 MultiSphere Design

In order to describe all possible solutions, alternatively to the “traditional” SD tree, MultiSphere introduces the concept of a *Tree of Promise* where the symbols constituting a candidate vector solution (i.e., the SD tree nodes) are described by their relative ordered distance to the received signal. Then, MultiSphere introduces a new *Tree of Promise partitioning method*, which adjusts to the transmission channel without compromising the ML optimality (see Section 3.1). The *Tree of Promise* (and therefore the original SD tree) is split into subtrees which are processed in parallel by the PEs. This partitioning can take place offline, based on the average channel characteristics, or “on-the-fly”, whenever the transmission channel changes following each QR decomposition. This adds preprocessing latency to that of the QR decomposition. However, the partitioning latency scales linearly with n_t in contrast to the QR decomposition latency which scales almost cubicly with the number of transmit antennae. After SD partitioning, MultiSphere applies a new *symbol-to-subtree allocation method* (see Section 3.1.3) which, in contrast to other schemes [18], maps nodes to PEs without introducing dependencies and minimizes redundant calculations across PEs. Each PE performs depth-first subtree traversal with Schnorr-Euchner enumeration [26], according to which, nodes are visited in ascending order of their PDs. Several approaches have been proposed [5], [8], to avoid exhaustive calculation and sorting of the PDs. However, they are not applicable to MultiSphere since their ordering is sequential (finding the kth smallest PD, requires finding the $(k-1)$th smallest PD first, starting from $k = 1$). To that end, in Section 3.2.1 we introduce a new *tree traversal and enumeration method* which meets MultiSphere's needs.

MultiSphere's tree searches execute nearly independently. They interact only once, after they have all reached their first leaf node. The r^2 of each subtree is then replaced by that of the leaf node with the minimum PD across all parallel SDs. The search is terminated when all subtrees have been searched, the detection output being the leaf node with the minimum PD across all subtrees. The overall processing latency is determined by the slowest parallel SD.

3.1 MultiSphere's Preprocessing: SD Tree Partitioning

MultiSphere's SD partitioning consists of a) the *seeds identification*, which finds N_{PE} paths (seeds) in the *Tree of Promise*, the ones most promising to constitute the correct solution (i.e., to be the transmitted vector) with N_{PE} being the number of available PEs and b) the *seeds to subtrees expansion*, which assembles subtrees around these seeds so that their union forms the original SD tree.

3.1.1 Seeds Identification

The *relative position vector* (RPV) $\mathbf{m}$ describes a tree path, in the *Tree of Promise* by means of the ordered (in terms of PDs) position of its nodes to the received observable. If the lth element of $\mathbf{m}$ equals k, then, for the corresponding path, its node at level l is the kth closest node to the received $\widetilde{y}_l$. If, for example, $\mathbf{m} = [1, 2, 3]^T$ the path consists of the node with the third smallest PD at the highest level of the tree (i.e., $\mathbf{m}(3) = 3$), its child with the second smallest PD at the second level of the tree, and its child with the smallest PD at the lowest level of the SD tree.

Finding the exact probability for each path to include the correct solution is clearly an non-trivial task that would require difficult integrations with no obvious, closed-form solutions. In order to avoid such calculations we propose a Metric of Promise (MoP) $\mathcal{M}$, related to a proposed approximation of the corresponding probability (please refer to the Appendix for details). In particular, the proposed MoP for an SD tree path with RPV $\mathbf{m}$ is

$$\mathcal{M}(\mathbf{m}) = -\sum_{l=1}^{n_t} \ln\left\{ e^{-\frac{\alpha_l [m_l - 1] |R_{ll}|^2}{2\sigma^2}} - e^{-\frac{\alpha_l m_l |R_{ll}|^2}{2\sigma^2}} \right\}, \quad (1)$$

with $\mathcal{M}(\mathbf{m}) \approx -\ln\{P[\mathbf{x_m} = \mathbf{s}^{(t)}]\}$, where $\mathbf{x_m}$ denotes the symbol vector related to path $\mathbf{m}$ and α_l depends on the minimum distance d_{QAM} between QAM symbols at level l (e.g., for $d_{QAM} = 2$, $\alpha_l = 1.11$). Then, the smaller $\mathcal{M}(\mathbf{m})$ is, the more likely it is for path $\mathbf{m}$ to include the correct solution.

The MoP in Eq. (1) requires knowledge of the noise variance σ^2. However, as shown in the Appendix, if σ^2 is not known, we can instead use the following simplified MoP

$$\mathcal{M}_s(\mathbf{m}) = \sum_{l=1}^{n_t} \alpha_l [m_l - 1] |R_{ll}|^2, \quad (2)$$

where, provided that the same QAM constellation is used per transmit antenna, the terms α_l will be the same for all l values, and can be ignored as they won't affect finding the paths with the smallest $\mathcal{M}_s$ values. As we show in Section 5, $\mathcal{M}_s$ is equally efficient with $\mathcal{M}$ regarding its ability to reduce decoding latency. However, the lack of knowledge of σ^2 prevents from using methods similar to the one proposed in Sections 3.2.2 or 3.2.3 to avoid the unnecessary allocation of PEs or to efficiently allocate (schedule) PEs in multicarrier systems. We note that both metrics are independent of any channel statistics, and independently exploit each channel realization. The metric $\mathcal{M}_s$ is an improved yet less complex version, of the heuristic MoP proposed in our original work in [1]. Since the MoPs are not a function of the actual received signal they can be pre-calculated before data detection (i.e., before sphere decoding). In addition, both $\mathcal{M}$ and $\mathcal{M}_s$ can be calculated in a recursive manner, similarly to traditional SDs. As we explain in the end of this Section, the seeds identification does not need to be exact to preserve ML optimality. Thus, they can be found in a K-Best manner with $K = N_{PE}$, requiring latency of the order of $N_{PE} \cdot n_t$.

Algorithm 1. MultiSphere's Seeds to Subtrees Expansion

```
MultiSphere- Seeds to subtrees expansion
Input: m, $|\mathcal{O}|$, $n_t$, $N_{PE}$, $i$                          // Seeds
Output: $T_i$                                                   // Subtree $i$
Initialize: $l \leftarrow n_t$, $T_i \leftarrow \emptyset$                        // Current level
Initialize: $\mathcal{B}_{(n_t+1)} \leftarrow \{1,\ldots,N_{PE}\} \cap [i-1, i+1]$         //
     $\mathcal{B}$ : $(n_t+1) \times N_{PE}$ non-unique indices buffer
 1: for $l = n_t$ to 1 do
 2:     if $\exists\ k,\ k \in \mathcal{B}_{(l+1)} :\ m_{i,l} = m_{k,l},\ \forall\ k \neq i$ then
 3:     $\mathcal{B}_l \leftarrow k$                         // Store non-unique indices
 4:     else break end if
 5: end for
 6: splitlevel $\leftarrow l$
 7: for $l =$ splitlevel to $n_t$ do
 8:     find $max(m_{k,l}),\ \forall\ k \in \mathcal{B}_{(l+1)}$
 9:     if $m_{i,l} = max(m_{k,l}) - 1$ then
        $T_i \leftarrow T_i \cup$ descendants and ancestors of $m_{i,l}$
10:     break
11:     else if $m_{i,l} < max(m_{k,l})$ then $bound(l) \leftarrow m_{(i+1),l} - 1$
        $T_i \leftarrow T_i \cup$ all nodes at $l$ with indices $j$ : $j \in [m_{i,l}, bound(l)]$,
        descendants and ancestors
12:     break
13:     else $bound(l) \leftarrow |\mathcal{O}|$
        $T_i \leftarrow T_i \cup$ all nodes at $l$ with indices $j$ : $j \in [m_{i,l}, bound(l)]$
        and their descendants
14:     end if
15: end for
16: return $T_i$
```

3.1.2 Seeds to Subtrees Expansion

The *Seeds Identification* process outputs N_{PE} seeds, each with its own RPV $\mathbf{m}_i$ ($i = 1, \ldots, N_{PE}$) for each of which there is one node defined per tree level. The subtrees expansion of a seed is then used to construct a corresponding subtree T_i (i.e., for each of which there is a range of nodes defined per tree level) which, similarly to the seeds, is also expressed in terms of RPVs. These subtrees will later be processed by the respective PE. Seeds $\mathbf{m}_i$ are sorted in ascending order of their indices $\mathbf{m}_i(l)$, starting from $l = n_t$. Seeds with the same elements from n_t until a level l are sorted in ascending order of elements at level $l-1$. Then, for each $\mathbf{m}_i$ the subtrees are created according to Algorithm 1, so that the union of all T_i forms the original SD tree, in order to preserve ML optimality. In the example of Fig. 1, the expansion of T_2 starts from

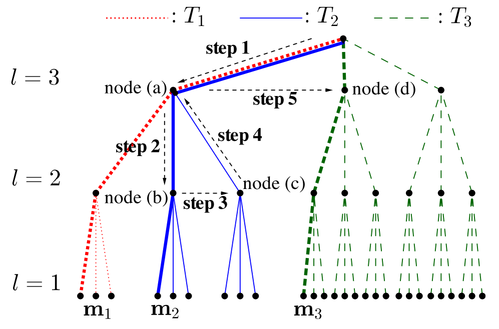

Fig. 1. Tree of Promise and expansion of seeds $\mathbf{m}_1 = [1,1,1]^T$, $\mathbf{m}_2 = [1,2,1]^T$, $\mathbf{m}_3 = [1,1,2]^T$, (bold) into subtrees T_1 (dotted), T_2 (solid), T_3 (dashed) in (ii). Their union is the full SD tree. Nodes may appear among several subtrees.

the first node that is part of $\mathbf{m}_2$ and not part of adjacent seeds, i.e., $\mathbf{m}_1$ and $\mathbf{m}_3$ (step 2). The algorithm continues by allocating sibling nodes and their descendants (step 3), and continues traversing up the tree and allocating sibling nodes with greater indices that do not belong to the following seed (step 5 allocates no such nodes). The algorithm will, in the worst case, reach level $l = 1$, and then traverse up to $l = n_t$ allocating nodes in the process, and resulting in a latency of $O(2n_t)$.

3.1.3 Symbol-to-Subtree Allocation

Tree partitioning provides the nodes to be processed by each MultiSphere SD as a function of their ordered distance. In Fig. 1, for example, subtree T_3, will consist of the 2^{nd} and 3^{rd} closest symbols at $l = 3$ (or the nodes for which $\mathbf{m}_3(3) = 2$ and $\mathbf{m}_3(3) = 3$) and all their descendants. In high order systems, finding the actual symbols would require exhaustive PD calculations and sorting of the corresponding nodes multiple times across the parallel SDs. To avoid these redundant calculations, MultiSphere uses an approximate predefined order to allocate symbols to subtrees, based on calculating *minimum euclidean distances* depending on the *relative* position of the received point and the constellation geometry. In addition, it uses a symbol mapping of *two-dimensional zigzag coordinates*. In one-dimensional symbol constellations, the sorted order of the symbols in terms of their distance to the received point can be easily found in a zigzag manner [5], [8] after finding the closest constellation symbol $s_l^{(c)}$ to the "equivalent (in the constellation domain) received point"

$$\overline{y}_l = \begin{cases} (\widetilde{y}_l - \sum_{j=l+1}^{n_t} R_{lj}s_j)R_{ll}^{-1}, & 1 \leq l < n_t \\ \widetilde{y}_{n_t} R_{n_t n_t}^{-1}, & l = n_t \end{cases}. \tag{3}$$

Using the zigzag concept each symbol in a two-dimensional constellation can be mapped in terms of its zigzag coordinates (zz_x, zz_y), as shown in Fig. 2. MultiSphere's preordering is based on the relative position of $\overline{y}_l$ to $s_l^{(c)}$. In particular, after finding $s_l^{(c)}$, we know that $\overline{y}_l$ (shaded triangle in Fig. 2) will always lay in a square with one of its edges at $s_l^{(c)}$ and a side length equal to the half of the minimum distance d_{QAM} between QAM symbols. Thus, a minimum euclidean distance d_{min} from any constellation point s to $\overline{y}_l$ can be calculated as in Fig. 2. Then, MultiSphere uses a predefined order that approximates the actual one. In particular, if we want to allocate to a subtree the kth closest symbol to $s_l^{(c)}$, MultiSphere allocates instead the symbol whose zigzag coordinates are the ones

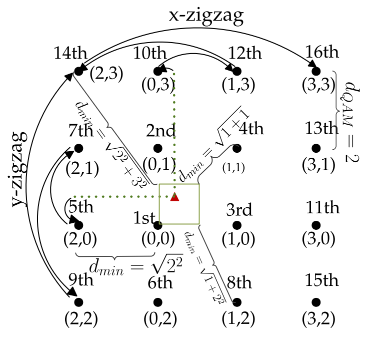

Fig. 2. Predefined order example for 16-QAM.

of the kth smallest d_{min}. For example, in Fig. 2, if the fourth closest symbol to $s_l^{(c)}$ needs to be allocated, we instead allocate the symbols with the zigzag coordinates (1,1). To precalculate the d_{min} values we assumed that $s_l^{(c)}$ is an inner constellation symbol. If two or more symbols have the same d_{min} but different zz_x or zz_y, the d_{min} values are sorted in an ascending order of their corresponding coordinates. We note that despite the fact that there are four possible squares where $\overline{y}_l$ can lie, all d_{min} values, predefined orders and corresponding zigzag coordinates exploit constellation symmetry, significantly reducing the architectural overhead compared to the preordering initially proposed in [1]. We note that this approximate ordering does not affect optimality since the union of all constructed sub-trees still forms the original SD tree. In addition, since the *relative* sequence is stored via zigzag coordinates, mapping is feasible even if $s_l^{(c)}$ is an outer constellation symbol. Then, d_{min} is still a valid lower limit of the corresponding euclidean distance, since zigzagging from $s_l^{(c)}$ will point to a symbol which is even further than what was initially assumed.

3.2 MultiSphere's Sphere Detection

3.2.1 Tree Traversal and Enumeration

Upon expanding a node, MultiSphere's SDs visit children nodes in ascending order of their PDs (see Section 2). To avoid calculating and sorting all PDs at each level, enumeration methods have been proposed [5], [8] which, as discussed, are not applicable to MultiSphere due to their sequential ordering. MultiSphere performs the following enumeration instead. From the set of potential symbols to be visited at a specific tree level and for each existing zz_x coordinate, we identify the symbols with the minimum zz_y. This results in a subset of at most $\sqrt{|\mathcal{O}|}$ symbols having unique zz_xs. Out of this subset, we calculate the PD of the symbol with the minimum zz_x for which $zz_y = 0$, if such a symbol exists. In addition, we calculate and store in a buffer $\mathcal{Q}$, of maximum size $\sqrt{|\mathcal{O}|}$ the PDs of the symbols with $zz_y \neq 0$. We first visit (and remove from $\mathcal{Q}$) the symbol with the smallest PD in $\mathcal{Q}$. If (zz_x^k, zz_y^k) are the zigzag coordinates of the kth symbol removed from $\mathcal{Q}$, then to find the next symbol, we calculate (and store in $\mathcal{Q}$) the PD of the symbol with $(zz_x^k, zz_y^k + 1)$. If $zz_y^k = 0$ then we also compute the symbol with $(zz_x^k + 1, zz_y^k)$. Fig. 2 for example shows that if the SD partition requires examining the 12th to the 16th closest symbols, then symbols $(1,3)$, $(2,3)$ and $(3,1)$ are first stored in $\mathcal{Q}$. Since $(1,3)$ has the smallest PD in $\mathcal{Q}$, no new symbol is added as $(1,4)$ does not exist. We then visit the symbol with the second smallest PD in $\mathcal{Q}$, i.e., $(3,1)$. Subsequently, we add $(3,2)$ to $\mathcal{Q}$ and the process continues.

As verified in Section 5 by both our software and VLSI design evaluations, MultiSphere can preserve the ML optimality due to three reasons: (a) the "Seeds to subtrees" expansion is such that, independently of the seeds, the union of all parallel subtrees will include all the nodes of the initial (sequential) SD tree. As a result, no possible solution is excluded from the search. (b) While the mapping of nodes to subtrees is approximate, no node is excluded from the final search; some nodes may instead be mapped onto different subtrees. (c) The new enumeration approach ensures that each parallel tree search visits nodes in ascending PD order as in the Schnorr-Euchner enumeration, which also preserves the ML optimality.

3.2.2 Adjusting the Number of Allocated PEs

Depending on the channel condition and SNR, allocating more PEs to an SD tree search can possibly increase complexity without any further latency reduction. In order to achieve low latency without unnecessary PE utilization, MultiSphere can allocate to an SD search only K PEs, K being the minimum value for which

$$\sum_{k=1}^{K} e^{-\mathcal{M}_k} \geq \beta, \tag{4}$$

with $\mathcal{M}_k$ being the kth smallest MoP. *Subtrees expansion* can then take place by using only K out of the N_{PE} available PEs, and their corresponding K seeds. The sum in (4) approximates the probability that the correct solution lies among these K seeds. Therefore, when this probability reaches a predefined value β that we evaluate via simulations in Section 5.1 then no more PEs are employed. Fig. 8 for example shows that by setting $\beta = 0.5$ at 16 dB SNR, allows utilizing 62 PEs to achieve the same average latency as a 128-PE MultiSphere at half of the latter's complexity.

3.2.3 PE Scheduling for MIMO Multicarrier Systems

In MIMO multicarrier systems, we can inherently parallelize the workload by allocating one sequential SD per subcarrier. However, and as we show in Section 5.1, such a method is inefficient when targeting the exact ML solution since the latency of the multi-carrier frame is determined by the "slowest" SD. As shown in Fig. 9, if we instead use MultiSphere with 8-PEs to parallelize the detection of each subcarrier and process each subcarrier sequentially, we reduce latency by a factor of three compared to allocating one sequential SD per subcarrier (52 in total). However, as described in Section 3.2.2, when a large number of PEs is available to MultiSphere, allocating all PEs to a subcarrier may unnecessarily waste processing power. To efficiently use the available PEs, we hereby propose a PE scheduling approach according to which MultiSphere processes the several subcarriers sequentially, starting from the one demanding the most PEs. If Eq. (4) is unfulfilled for K PEs with $K < N_{PE}$ we process the subcarrier using all available PEs. If, on the other hand, (4) is met, still leaving enough available PEs for more subcarriers to fulfill (4) for their corresponding MIMO channel, we can then process these subcarriers in parallel. As shown in Fig. 10, this allows for efficient PE utilization when there can be many more PEs than subcarriers and reduces latency by several orders of magnitude compared to per-subcarrier parallelization strategies.

3.2.4 Approximate, Fixed-Complexity MultiSphere

MultiSphere can provide the exact ML solution at substantially reduced processing latency compared to sequential SDs. This latency, though, can significantly vary with the SNR and channel condition. Solutions like the FSD [23] or breadth-first SDs [13], [27], [28] provide a fixed and predetermined processing latency by sacrificing ML optimality. In the same manner, MultiSphere can be terminated at any time instant, after each SD finds its first candidate solution. In such a case, MultiSphere's latency can be flexibly set to any value at runtime, in contrast to traditional, approximate breadth-first approaches which can only set latency at design time. Consequently, MultiSphere allows for efficient trade-offs between error-rate performance, latency and N_{PE} for a given transmission scenario.

From this family of approximate SDs, as discussed in Section 1, FSD enjoys akin parallelization properties with MultiSphere. However, while MultiSphere can use any number of PEs, the FSD requires N_{PE} to be an integer power of $|\mathcal{O}|$. In addition, the FSD determines the tree paths to run in parallel in a pre-defined manner and cannot adjust to the transmission conditions. In Section 5.1, we compare the FSD against an approximate, minimum latency version of MultiSphere (denoted as a-MultiSphere) which only visits the N_{PE} most promising paths (i.e., *seeds*) and the SD's output is the seed with the minimum euclidean distance. This suboptimal approach requires neither *MultiSphere's Seeds to Subtrees Expansion* (Section 3.1.2) nor *MultiSphere's Tree Traversal and Enumeration* (Section 3.2.1) enabling high-throughput designs with a large number of information streams. In Section 5.1 we show that for a 16×16 MIMO configuration, a-MultiSphere performs similar to the FSD with one eighth of the latter's PEs, while our VLSI post-synthesis evaluation in Section 5.2 shows that this advantage translates to almost an order of magnitude higher hardware efficiency.

In soft-output systems, calculating the soft information requires calculating multiple constrained ML problems [29] and therefore, MultiSphere is still applicable. However, the complexity of traditional soft-output SD approaches [29], [30], [31] becomes impractical when applied to large MIMO systems and their parallelization would thus require an impractically large number of PEs. To the best of our knowledge, the only soft detection solution that is applicable to large MIMO systems is the soft-output version of the FSD (hereafter referred to as SFSD). Therefore, for soft-output systems we focus on the approximate version of MultiSphere and compare its performance against the SFSD and the recent, partial marginalization-based approach of [32]. In Sections 5.1 and 5.2 we show that in a similar 16×16 MIMO scenario and at 0.75 code rate, the soft-output version of a-MultiSphere achieves a performance advantage that is consistent with that of the hard-output version.

4 MultiSphere: VLSI Architecture

The primary challenge in retaining MultiSphere's algorithmic advantage in practice lies in the design of a VLSI architecture which efficiently addresses MultiSphere's tree traversal and

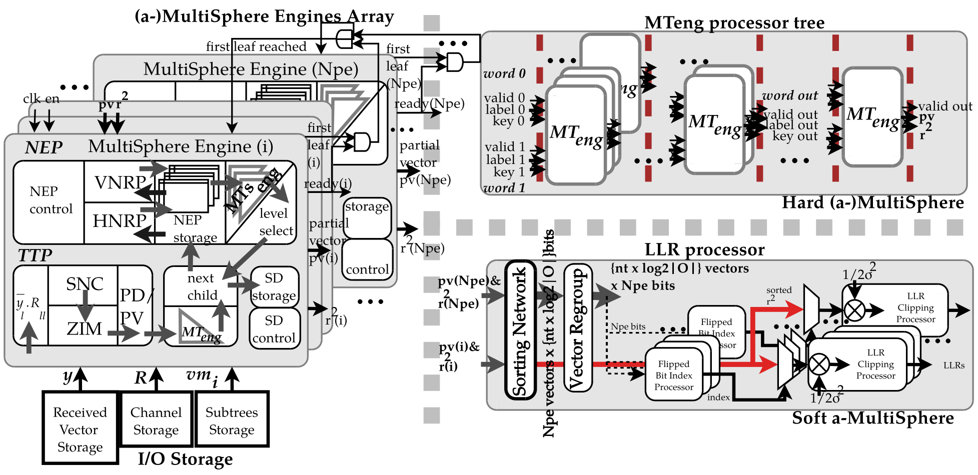

Fig. 3. MultiSphere's Sphere Detection engines and parallel framework.

enumeration (Section 3.2.1). Of particular challenge is first, finding the subset of at most $\sqrt{|\mathcal{O}|}$ nodes which may be visited at a specific tree level without any taxing sorting operations, and second, selecting the next sibling node within the subtree T_i by avoiding sequential or high-complexity comparisons. In the following, we present the design of a VLSI architecture which efficiently tackles both challenges at a minimal area and critical path overhead. We note that our architecture serves as a proof-of-concept of MultiSphere's principles when all PEs are allocated to a single subcarrier. As we will show in our evaluation, dynamic allocation of PEs to subcarriers is suitable when many PEs are available, necessitating algorithmic co-design and tradeoff study of efficient dispatchers and interconnection networks and is thus left for future work.

4.1 Parallel MultiSphere Detection Engines

MultiSphere's processing approach maps to the single-instruction multiple data paradigm, i.e., tree-search is performed in parallel on multiple T_is. Moreover, MultiSphere's tree partitioning (Section 3.1) allows for detection with minimal dependencies and data exchange, as during detection only the radius is exchanged, only once and only in the exact ML case. To minimize processing latency and to allow evaluation with arbitrary PEs, our framework is based on distributed memory storage and de-centralized control units. Fig. 3 depicts the overview of MultiSphere's detector as well as the parallel engines' architecture, interconnecting an arbitrary number of detection engines with minimal overhead. The $\mathbf{R}$ matrices and the $\mathbf{y}$ values are broadcast to all engines, while T_is in the form of validity matrices vm are assigned to each engine. Exact engines signify having reached the first leaf on their T_is by a distinct signal which is also used as a self-disabling latched pulse. All PD values along with their partial vectors (i.e., keys and labels respectively) are processed by comparator networks arranged in a binary tree fashion, i.e., processor arrays denoted as *MinTrees* (MT_{eng}) which in essence serve as the engines' interconnection network. The network's output i.e., the final partial vector and partial distance is either broadcast to all detection engines (exact case) or forwarded to the output (approximate case). When all exact engines reach their first leaf, they store the output corresponding to the minimum PD, the self-disabling signal is reset and they resume on the next clock cycle. In the case of instantiating many detection engines, the critical path and fanout can be reduced through input storage replication. Despite MultiSphere's concurrency, exact-ML depth-first detection is stochastic and one cannot assume that the PEs will finish in a sequential order. Thus, employing the efficient reduction circuits in [33], [34] would increase latency, particularly in the high SNR range. In the case of the soft-output a-MultiSphere, each engine's partial vector and partial distance pair are concatenated onto a single vector and forwarded to an *LLR processor* (Fig. 3) which calculates the Log-likelihood ratios and whose architecture we describe in the following section.

4.2 The MultiSphere Sphere Detector Engine

Exact Multisphere. In this section, we describe the internal design of the exact MultiSphere SD engine, which is based on a generalization of the folded one-node-per-cycle architecture first defined in [8] with additional node replacement and enumeration logic, following similar principles as [8]-ASIC-II. Despite the various SD approaches [8], [13], [18], [28], [35], [36], [37], this is, to the best of our knowledge, the most efficient design for depth-first SDs that can find the exact ML solution while visiting one node per clock cycle. We employ integer arithmetic and present a VLSI architecture which enables MultiSphere's efficiency via bit-parallel operations also supporting traditional decoding and is also scalable to denser constellations as our evaluation shows.

4.2.1 Tree-Traversal Processor (TTP)

The *TTP* (Fig. 4-left) selects $\sqrt{|\mathcal{O}|}$ nodes to be visited on the current tree level l and computes their PDs. To avoid dividing by R_{ll}, (Eq. (3)), we multiply all constellation point values by R_{ll}. The processing datapath of a-MultiSphere's branch (Fig. 4, middle) is almost identical MultiSphere's *TTP* excluding the node collector described below.

Low-Complexity Multiple Constant Multipliers (MCMs). Due to the large number of multiplications required for detection, designing efficient constant coefficient multipliers is critical. Our designs employ a multiplier which stores only the positive integer constellation values in a $\left[\frac{\sqrt{|\mathcal{O}|}}{2} \cdot (\log_2(\sqrt{|\mathcal{O}|}) + 1)\right]$-bit lookup table (Fig. 4-right). In order to map constellation points, we employ binary indices that also address the lookup table while the most significant

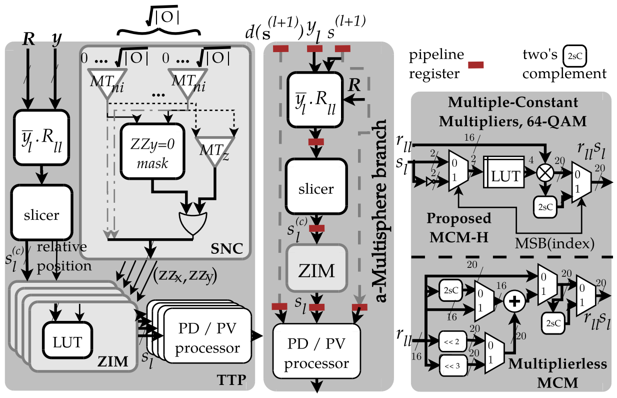

Fig. 4. MultiSphere's *Tree-Traversal Processor* (left) illustrating task parallelism for finding the closest symbol and collecting the nodes within T_i. Consequently, the subtree node collector unit and *Zigzag to Index Mappers* incur a minimal overhead to the critical path. The a-MultiSphere branch (middle): pipeline registers are depicted via red parallelograms. Integer MCMs (right): multiplier-based and multiplierless.

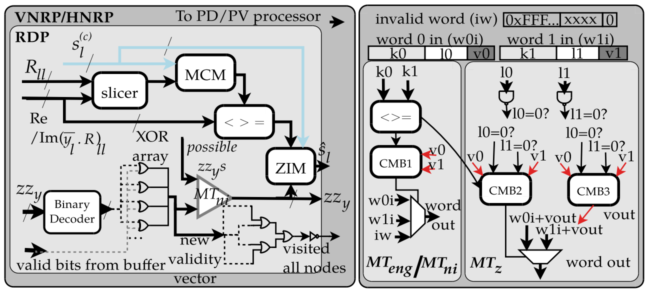

Fig. 5. Architecture of MultiSphere's Node Replacement Processors (left) realizing enumeration, PD calculation and partial vector generation. Right side: *MinTree* processors' architecture.

binary index bit adjusts both the multiplication result and the final lookup address (for negative constellation values.) We also considered two additional multipliers: a) in which we store positive and negative integer constellation points in a $(\sqrt{|\mathcal{O}|} \cdot (\log_2(\sqrt{|\mathcal{O}|}) + 1))$-bit lookup table (full-depth, denoted as *MCM-F*), and b) the flexible multiplierless approach of [20] for up to 64-QAM. Due to the exact ML nature of MultiSphere, we employ two's complement arithmetic (i.e., "2sC" in Fig. 4-right) instead of negation via NOT gates as in [20]. In Section 5.2 we present a thorough efficiency evaluation of multipliers via synthesis.

Slicer. Following the computation of $\overline{y} \cdot R_{ll}$, the engine determines $s_l^{(c)}$ via its slicer. To employ the aforementioned low-complexity multipliers, slicing relies on intermediate integer boundaries scaled by R_{ll}. We then compare the received symbol with the scaled boundary and directly map it to a constellation point index. Additionally, we detect the received symbol's relative position (i.e., left or right) to the selected closest point for subsequent employment by our "*Zigzag to index Mapper*". We determine the relative position via two comparators and one "2sC" module for 16-QAM (six comparators and three "2sC" modules for 64-QAM).

Subtree Node Collector (SNC). One of MultiSphere's main architectural novelties, the *SNC* collects at most $\sqrt{|\mathcal{O}|}$ nodes which the detector will visit on a particular level, by efficiently avoiding complicated, sorting operations which would adversely affect the critical path. Our proposed approach is based on the MT$_{ni}$ comparison processors (Figs. 4 and 5) which operate in parallel to the $\overline{y} \cdot R_{ll}$ calculation and the *Slicer*. To define and store the T_i for each engine, we consider the nodes per level arranged in ascending order of their zigzag coordinates, first by zz_x and then by zz_y, i.e., $(0,0)\ (0,1) \ldots (\sqrt{|\mathcal{O}|}, \sqrt{|\mathcal{O}|})$. In this manner, we only need a single bit to denote a node's presence in T_i and therefore defining the latter in a detection engine through a *validity matrix* (*vm*) requires $n_t \cdot |\mathcal{O}|$ bits. For every unique zz_x, we employ a single MT_{ni} processor which outputs the minimum valid zz_y. Note that the arrangement of the MT_{ni} processors is a function of the constellation size and thus the actual zz_xs need not be processed by the MT_{ni} network. Thus, the MT_{ni} comparators' width is just $\log_2\sqrt{|\mathcal{O}|}$ bits. Next, we generate a binary mask of $\sqrt{|\mathcal{O}|}$ bits which signifies (via zeroes) the nodes (among the $\sqrt{|\mathcal{O}|}$ chosen) for which $zz_y = 0$. In parallel, a single, more complex MT_z processor processes both coordinates to denote the minimum zz_x for which $zz_y = 0$ (Fig. 4). We then decode the index into a $\sqrt{|\mathcal{O}|}$-bit vector and *OR* the result with the mask to get the final valid nodes.

Zigzag to Index Mapper (ZIM). This mapper employs the closest point index, the single-bit relative position of the received symbol (left or right) and the zigzag coordinates from the *SNC* to generate the indices of the nodes selected at the current level. Mapping is based on an optimized lookup table to maintain a low area and critical path overhead.

4.2.2 Node Enumeration Processor (NEP)

The *NEP* determines the next sibling and stores the current search state per tree level (i.e., the node attributes). The state normally consists of the node's partial vector $\mathbf{s}^{(l)}$, its PD, and a single bit flag which verifies validity. In the case of MultiSphere, the proposed 2D enumeration (Section 3.2.1) requires additional storage of zz_x, zz_y per node. The current tree-search state also involves storing the euclidean distance $d(\mathbf{s}^{(l+1)})$ and $\overline{y} \cdot R_{ll}$, both of which the *TTP* has already computed. Each storage unit, organized as a register bank, stores $\sqrt{|\mathcal{O}|}$ elements of: a) $\log_2\sqrt{|\mathcal{O}|}$ bits for the partial vectors, b) parameterized width for the PDs, c) single bits for validity and d) $2 \cdot \frac{\log_2|\mathcal{O}|}{2}$ bits for zz_x, zz_y. Additionally, we employ $|\mathcal{O}|$ bits per level to store the level's current validity matrix $vm(l)$.

Node Storage. Our proposed storage solution adopts a parallel load/store register file. To reduce switching activity, we employ fine-grained clock enabling on all SD registers. While MultiSphere traverses down the tree, its control unit ensures through a level write signal that only a single storage unit will be active. We notice (Fig. 2) that the order of the node indices per level depends on the received symbol (i.e., we cannot directly map the node's constellation index to its location in the buffer). We thus address our storage via zz_x to avoid expensive permutations.

Node Replacement Processors. Depending on zz_y, MultiSphere can replace each node with up to two siblings (Section 3.2.1). To efficiently find the replacements and maintain the one-node-per-cycle processing rate, we introduce two distinct modules of similar functionality, the *Horizontal Node Replacement Processor* and the *Vertical Node Replacement Processor* (*HNRP* and *VNRP*, Fig. 5). They consist of the *Replacement Discovery Processor* (*RDP*) that computes the attributes of a sibling and forwards these attributes to the PD / partial vector processor. The *RDP*'s *slicer* and multiple constant multiplier

(MCM) recompute the closest node's indices and their relative position to the received symbol.1 As MultiSphere explores potentially a subset of the constellation at each T_i level, incrementing zz_y by one i.e., a simple vertical zigzag and then sequentially checking if each resulting node is part of T_i (or the constellation), is indeed one solution albeit an inefficient one. We instead employ a $\sqrt{|\mathcal{O}|}$-bit vector (*valid bits from buffer* in Fig. 5) corresponding to the current subconstellation state from the validity matrix $vm(l)$ as selected via zz_x. This stored part of the array is *XOR*-ed with the binary decoded zz_y and the result is used as: a) a validity vector for the *Replacement Discovery Processor*'s MT_{ni} to select the minimum among all remaining valid zz_ys and b) the *new validity vector* which will replace the corresponding part of $vm(l)$. Once all subconstellation nodes have been visited, the *new validity vector* consists of zeros and the *visited all nodes* signal is generated. The SD will then avoid storing the *VNRP*'s result, invalidate the buffer location corresponding to the selected subconstellation and bypass PD calculation. Similarly, the *HNRP*'s *valid bits from buffer* signal denotes instead the remaining valid nodes for which $zz_y = 0$. To avoid storage conflicts, only the *VNRP* invalidates buffer locations. Notice that now the *VNRP* and *HNRP* node attribute outputs have to be written to the storage unit's register file. The outputs' real indices are by definition different and this is exploited to establish conflict-free storage access. The node attribute vectors generated by each replacement unit are de-multiplexed into a specific location of the $\sqrt{|\mathcal{O}|}$-element register file and the two de-multiplexed vectors are then merged into one by an *OR* operation. The merged vector is then employed for storage. When the replacement is invalid, decoding of the index is disabled and nothing gets stored.

Approximate Multisphere (a-Multisphere): a-MultiSphere's fixed processing complexity allows for a pipelined parallel VLSI architecture where we consider a processing element as the fully-instantiated logic required to process a single SD path across all tree levels. a-MultiSphere's processing datapath is almost identical to that of MultiSphere's (Fig. 4) *TTP* except for the *Subtree Node Collector*. Compared with the FSD, a-MultiSphere incurs a minor overhead, mainly involving the *ZIM* and the additional pipeline. In the $n_t = 16$ case, a-MultiSphere's initial latency is $219 + \log_2(N_{PE})$ clock cycles, corresponding to a pipelined MT_{eng} array.

MinTree (MT) Processor Trees: MultiSphere's VLSI architecture relies on trees of *MinTree* processors (Fig. 5): a) MT_{ni} that output the minimum valid zz_y per zz_x, employed also inside the *RDP*, b) MT_z that output the minimum zz_x for which $zz_y = 0$, and c) MT_{eng} interconnecting the engines and used within each detector. Fig. 5-right shows the architecture of each *MinTree* processor. Each processor outputs a valid word containing the actual value (*key*), the corresponding rank (*label*) of each key and its validity bit, based on combinational (*CMB*) and comparison ($< > =$) circuits. Notice that MT_{eng} and MT_{ni} are almost identical apart from the unused label in the latter. In Section 5.2 we assess the complexity of each processor, its contribution in the total cost of MultiSphere and in tree arrangements.

LLR Processor. For calculating the log-likelihood ratios, instead of the MT_{eng} processor tree, the partial vectors and distances are initially processed by a streaming parallel bitonic sorting network (i.e., *SN1* in [38]), modified to wholly or partly employ each vector as the sorting key. We chose this particular architecture as it features the lowest latency while requiring the least amount of resources [38]. Apart from the sorted partial distances, the sorting network outputs N_{PE} partial vectors of $n_t \times \log_2|\mathcal{O}|$ bits, i.e., equal to the number of LLRs that need to be calculated. The partial vectors are then regrouped into $n_t \times \log_2|\mathcal{O}|$ vectors of N_{PE} bits each, where the first N_{PE}-bit vector contains the leftmost bit from each "sorted" partial vector (i.e., corresponding to the sorted partial distances) and so on. Hence, the leftmost bits (hereafter referred to as the ML bits) of the regrouped vectors correspond to the minimum partial distance. Each of these ML bits along with each regrouped N_{PE}-bit vector are input to a *Flipped Bit Index Processor* which computes the index of the first bit inside this vector that is non-equal to the ML bit. This selects one partial distance out of N_{PE}, which, along with the minimum partial distance and $\frac{1}{2\sigma^2}$ are used to compute the final LLR after clipping. We note that the proposed LLR processor can be flexibly (e.g., sorting network size, number of index and clipping processors) instantiated to meet specific device requirements; for the purpose of exploration, our evaluation assumes a fully parallel sorting network of up to N_{PE} keys and an expansion of up to $16 \times \log_2(64)$ LLRs.

5 Performance Evaluation

Here we jointly assess MultiSphere's, exact and approximate, algorithmic and VLSI architecture performance. MultiSphere's algorithmic performance is evaluated via simulations in terms of processing latency2 and overall complexity.3 For our architectural comparisons, we implement the software and VLSI versions of the sequential approach for which we replace the exhaustive SE enumeration with the PAM-based enumeration in [12] (hereafter referred to as our Sequential PAM SD) since it a) scales better with dense QAM constellations and b) expands a subset of $\sqrt{|\mathcal{O}|}$ nodes on each level, similarly to MultiSphere's bound (Section 3.2.1) Unless specifically stated otherwise, all euclidean distance calculations employ the exact l^2 norm. For consistency and fairness, we compare a-MultiSphere against our own flexible FSD VLSI architecture for hard and soft information-based detection. All of our VLSI architectures follow the design principles of Section 4. For our evaluations the seeds have been identified via a K-Best SD, with $K = N_{PE}$.

5.1 Algorithmic Performance Evaluation

We first evaluate MultiSphere's exact version in an uncoded, 16-QAM modulated 10×10 MIMO multi-carrier system assuming sorted QR decomposition (SQRD) as in [39]. We mathematically model each sub-channel between a

1. Note that the *TTP* has already computed these and they can be stored at the expense of increased area requirements in large antenna setups. This would slightly reduce node replacement delay only, as it lies outside the critical path (i.e., the *TTP*).

2. Via the number of visited nodes, assuming that one node is visited among those examined at every time instant [5], [8], [18], [20], [37].

3. Via the number of partial distance calculations performed, depending on the SD algorithm [5], [8], [12].

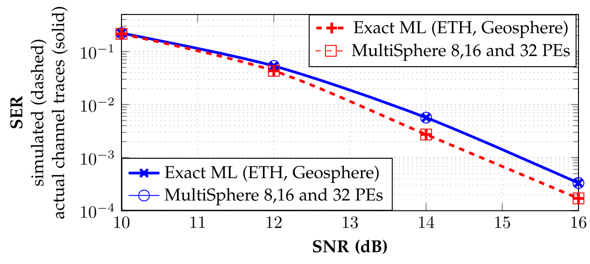

Fig. 6. MultiSphere's symbol error rate (SER) in actual and mathematically modelled channels (10×10, 16-QAM).

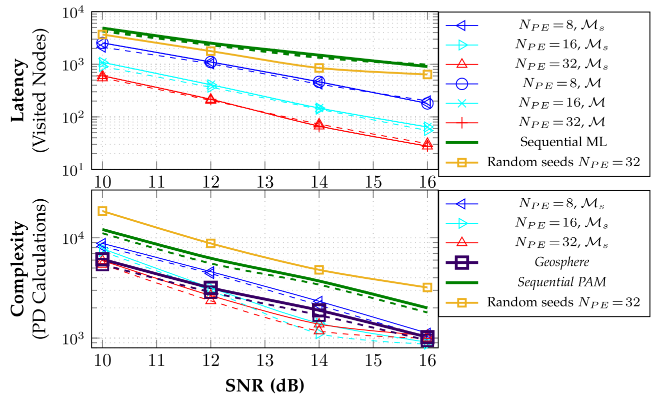

Fig. 7. MultiSphere's latency (top) and complexity (bottom) versus the *Sequential PAM* and *Geosphere* SDs for 10×10 16-QAM MIMO. We evaluate via both mathematically modelled channels (dashed lines) and actual channel traces (solid lines). Fig. 7-top considers both MoPs, while Fig. 7-bottom only MoP $\mathcal{M}_s$.

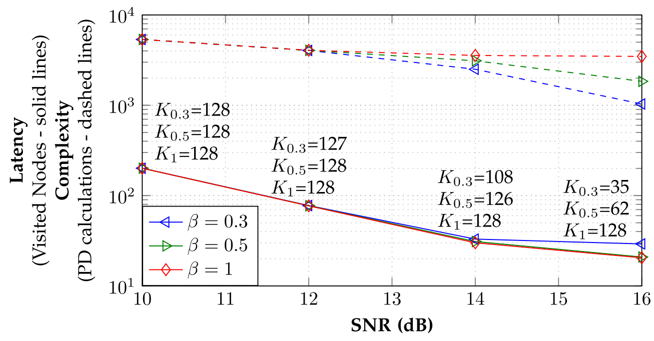

Fig. 8. MultiSphere when we adjust the number K of employed PEs ($N_{PE}=128$, 16-QAM 10×10 MIMO). Presented results involve the $\mathcal{M}$ MoP and actual channel traces. Similar results hold for mathematically modelled channels.

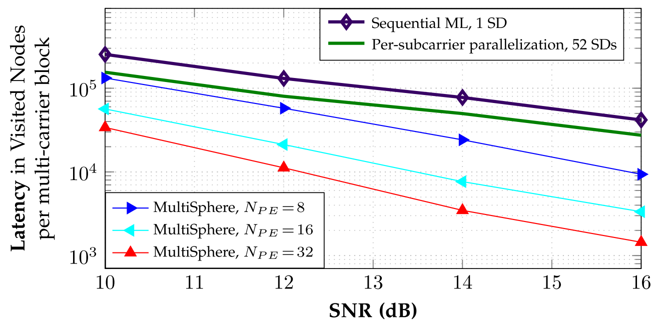

Fig. 9. MultiSphere's multi-carrier block latency versus per subcarrier parallelization (16-QAM, 10×10 MIMO, $N_{SC}=52$). Results shown for actual channel traces and $\mathcal{M}_s$. Similar results hold for mathematically modelled channels.

transmit-receive antenna pair as a 5 tap i.i.d. Rayleigh channel (in the time domain). We also evaluate MultiSphere via actual channel traces, measured[4] in indoor conditions. For our evaluations the channel is static over the transmission of a packet, i.e., it is assumed that one packet is transmitted per channel coherence time. As a result, preprocessing (i.e., channel estimation, QR decomposition, Seeds Identification) takes place once, at the beginning of each packet.

Single-Carrier Latency and Complexity. Fig. 6 verifies that MultiSphere's ML optimality in all cases. Fig. 7 depicts MultiSphere's latency and complexity for several N_{PE} cases in comparison with the state-of-the-art *Sequential PAM* and *Geosphere* [5] SDs, for both MoPs ($\mathcal{M}$ and $\mathcal{M}_s$, Section 3.1.1). We note that our algorithmic evaluation does not consider the latency overhead of finding and distributing the minimum r^2 across the N_{PE}s. Fig. 7 validates that MultiSphere can consistently decrease latency when N_{PE} increases; for $N_{PE} = 16$, MultiSphere reduces latency by more than an order of magnitude compared to sequential SDs. Moreover, as Fig. 7-bottom shows, MultiSphere reduces latency without substantially increasing complexity. In particular, the overall complexity can be even smaller than that of the highly-optimized, state-of-the-art sequential SDs examined. In addition, Fig. 7-top shows that both of the the proposed MoPs i.e., $\mathcal{M}$ and $\mathcal{M}_s$ attain a very similar latency reduction performance. Moreover, Fig. 7-top verifies that the latency advantage of MultiSphere is consistent for both mathematically modelled channels and actual channel traces. Fig. 7 also displays the latency and complexity for $N_{PE} = 32$, when, instead of using the proposed method (Section 3.1.1), we always include the most promising seed and randomly choose the rest. Compared to the sequential SD, this reduces latency only by approximately 20 percent though also increases complexity by 50 percent. For the same N_{PE}, our proposed *seeds identification* method reduces latency by 29× and has a lower complexity compared to the sequential SD.

Adjusting the Employed PEs. Fig. 8 highlights the efficiency of our method in Section 3.2.2 that adjusts the number of utilized PEs (K) and therefore complexity, as a function of β, without affecting the achievable latency. We note that for $\beta = 1$ all available PEs are used ($K_1 = N_{PE}$), which, as shown in Fig. 8, results in excessive overall complexity in the high SNR region, whereas adopting a very small β leads to underutilization of the available PEs and a processing latency "floor". By setting $\beta = 0.5$, a good trade-off between latency, complexity and number of utilized PEs is accomplished. Then, compared to allocating all PEs ($\beta = 1$), we can reduce complexity by 50 percent without a noticeable latency increase, and with the ML solution still being guaranteed. Via extensive simulations we have validated that the approach is insensitive to the exact selection of β. Setting β to 0.5 ± 0.1 practically leaves latency and complexity unaffected.

Multi-Carrier Performance and Scheduling. Figs. 9 and 10 compare MultiSphere to a straightforward scheme adopting per-subcarrier parallelization according to which, one exact,

4. A 20 MHz, 52 subcarriers MIMO-OFDM system was realized via WARPv3 radios and WARPLab software [40], in the 5 GHz ISM band. Channel traces for single antenna users from 10-antennae Access Points (APs) were measured separately and combined for each 10×10 channel realization. To emulate simple user selection strategies, the SNR of jointly scheduled users does not differ by more than 3 dB.

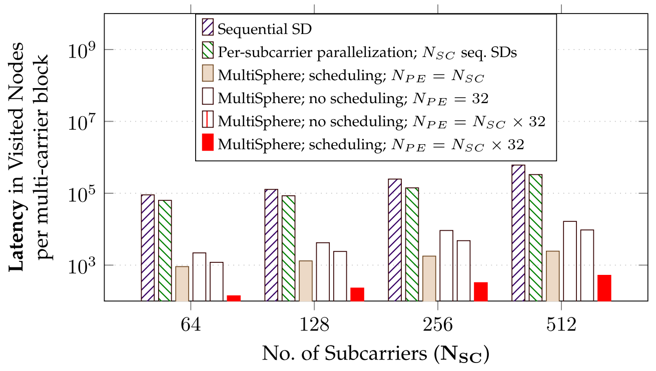

Fig. 10. MultiSphere's multi-carrier block latency v. PE/subcarrier processing with respect to N_{SC} (simulated channels, $\mathcal{M}$ MoP, 10×10 MIMO, 16-QAM, 16 dB SNR).

sequential SD processes each subcarrier. Fig. 9 displays results using actual channel traces consisting of 52 active subcarriers for the purpose of more realistic comparisons. We see that conventional, per-subcarrier parallelization, fails to efficiently reduce latency despite the large number of employed PEs (52), since the subcarrier with the highest latency determines the multi-carrier block's latency as well. On the other hand, sequentially processing the subcarriers via an 8-PE MultiSphere, results in a 3× latency reduction compared to per-subcarrier parallelization via sequential SDs (16 dB SNR). Moreover, in the high SNR range and with a 32-PE MultiSphere we reduce latency by more than an order of magnitude. Fig. 10 displays latency when targeting exact ML detection, for a varying number of subcarriers N_{SC}, and multiple configurations and N_{PE} values. We show that straightforward, per-subcarrier parallelization is incapable of efficiently reducing latency while preserving ML optimality. On the other hand, Fig. 10 shows that for the same degree of parallelism ($N_{PE} = N_{SC}$) MultiSphere combined with the PE scheduling of Section 3.2.3, and $\beta = 0.5$, can provide a latency reduction of more than two orders of magnitude, compared to per-subcarrier parallelization (for $N_{SC} = 512$). In addition, we show that MultiSphere, in combination with the proposed scheduling, is the first method that can exploit any number of PEs (e.g., $N_{SC} \times 32$) and consistently reduce processing latency, while still providing the exact ML solution. Without scheduling, even if the PEs are enough for MultiSphere to reach the minimum exact SD latency (i.e., $2n_t - 1$ nodes), due to the sequential

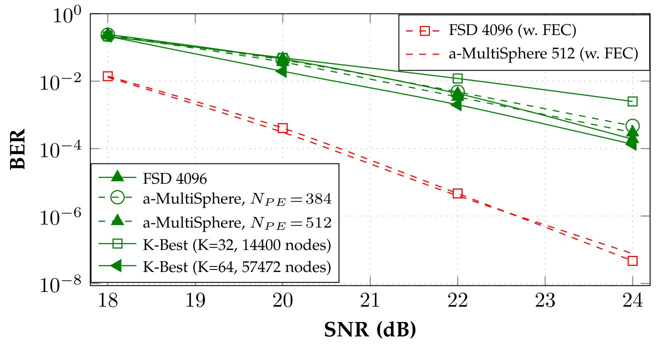

Fig. 11. Error rate of a-MultiSphere, FSD and K-best SDs in mathematically modelled channels for both uncoded and coded systems (64-QAM, 16×16 MIMO, 0.5 rate $(133/171)_8$ convolutional code) and varying N_{PE}.

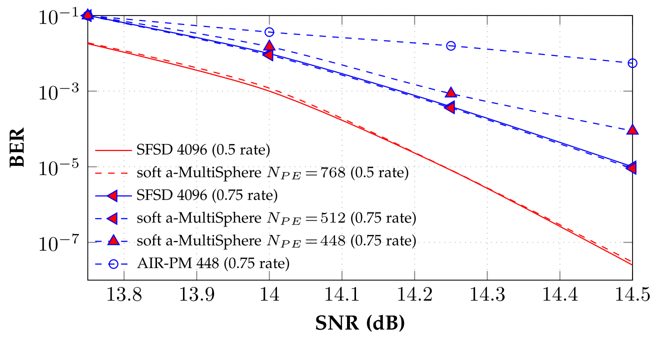

Fig. 12. Soft a-MultiSphere v. SFSD: BER performance with 0.5 and 0.75 rate LDPC channel code ($|\mathcal{O}| = 64$, 16×16 MIMO).

subcarrier processing, the overall minimum processing latency will be of the order of $\mathcal{O}(2 \cdot n_t \cdot N_{SC})$.

Approximate Detection. Fig. 11 compares a-MultiSphere (Section 3.2.4) with FSD and the K-best sphere decoder [13]. For fairness, a-MultiSphere, FSD and K-best decoder employ the sorted QR decomposition tailored to the FSD [23]. For the K-best detector we find the K-best siblings via a geometrically-based enumeration [28], [41]. Fig. 11 shows that since a-MultiSphere can focus its processing power on the most promising paths to include the correct solution, it consistently outperforms FSD for the same number of PEs, both for coded and uncoded systems, and with the gains increasing when smaller error-rates are targeted. Then, depending on the SNR, and for a 16×16, 64-QAM modulated system, a-MultiSphere can provide a similar error-rate performance to the FSD by utilizing less than one tenth of the PEs (384 instead of 4,096). In addition, as discussed in Section 1, FSD requires N_{PE} to be a multiple of the order of the transmitted constellation, which makes FSD inefficient for very-dense constellations. On the contrary, a-MultiSphere can efficiently utilize any number of available PEs. Notice that for SNRs of practical interest, the K-best SD can achieve a similar error-rate as a-MultiSphere, albeit at a much higher complexity premium i.e., requires 14,400 nodes, or equivalently, 900 a-MultiSphere paths (3,592 for $K = 64$).

MultiSphere for Soft-Output Systems. In Fig. 12 we compare the soft version of a-MultiSphere (where soft information is approximated by only exploiting the N_{PE} seeds), to the soft-output version of the FSD (i.e., SFSD) [42], and to the partial marginalization-based AIR-PM detector [32] for its minimum complexity, where one layer is fully expanded, and 448 paths are visited. To the best of our knowledge, these are the only approaches that can practically apply to large MIMO systems. Fig. 12 shows that, similarly to the hard-output case, the number of required PEs for a-MultiSphere to achieve the same error-rate performance with SFSD can be nearly an order of magnitude fewer. Moreover, for the same number of visited paths, MultiSphere substantially outperforms AIR-PM in terms of error-rate. In contrast to MultiSphere, AIR-PM is not as flexible, hence in order to further improve the latter's error-rate performance, an impractical number of approximately $3 \cdot 10^4$ nodes would need to be visited.

5.2 VLSI Architectures Evaluation

In order to explore scalability, we first assess the area and delay of the exact and a-MultiSphere PEs as well as the LLR processor's for $|\mathcal{O}| \in \{16, 64\}$-QAM modulation and

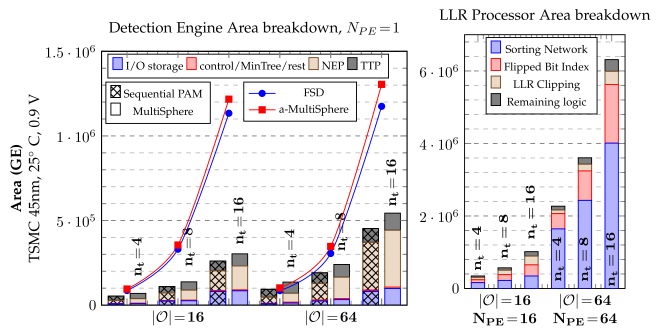

Fig. 13. Detection engine scalability and overhead with respect to n_t and $|\mathcal{O}|$: area (GE) breakdown at maximum frequency for MultiSphere, the *Sequential PAM* SD, a-MultiSphere and *FSD* (left plots). The right plots depict the LLR processor's scalability and overhead when setting $N_{PE}=|\mathcal{O}|$.

$n_t \in \{4, 8, 16\}$. We initially employ 24-bits for $d(\mathbf{s}^{(l)})$, 16-bits for $\mathbf{R}$ and 18-bits for the noise variance, in order to assess the worst case impact of scaling n_t and $|\mathcal{O}|$ (i.e., to reach ML performance at $n_t = 16,\ |\mathcal{O}| = 64$). Next, we jointly evaluate area requirements, performance and dynamic power consumption based on our algorithmic results (Section 5.1) using a 16-bit datapath for 16-QAM and retaining the 24-bit $d(\mathbf{s}^{(l)})$ for a-MultiSphere at 64-QAM. The detectors' highly modular and parametric Verilog RTL code is synthesized using the Synopsys Design Compiler and TSMC 45 nm standard cell libraries at 25°C and 0.9V. We apply the actual channel traces to simulate the gate level netlist and generate the corresponding switching activity files. We then estimate the worst-case (i.e., the channel changes with every new subcarrier) average dynamic power consumption for the SNR values of Section 5.1 using Synopsys' Power Compiler and respectively evaluate hardware and energy efficiency (denoted as H_{eff} and E_{eff}) via the bps/GE and Joules/bit (J/bit) figures of merit.[5] We also compare our post-synthesis results via technology scaling[6] with the exact, and the approximate $l_{\widetilde{\infty}}$ enumeration SDs in [43], the approximate method in [44] (which supports only up to QPSK) and the high-throughput SDs in [45], [46]. We note that our designs can instantiate arbitrary N_{PEs}; hitherto presented results are only indicative due to workstation memory limitations.[7]

Single-Engine Scalability. Here we show that MultiSphere's relative architectural overhead can be kept at low levels and decreases in large antenna setups (which constitute this work's main focus). We compare area requirements and maximum achievable frequency of the MultiSphere, Sequential PAM SD, a-MultiSphere and FSD PEs. Post-synthesis results in Fig. 13 show that the exact and a-MultiSphere's respective gate count overhead is reasonable at 26 and 10 percent for $n_t = 4$, reducing to 13.9 and to 7.3 percent in the $n_t = 16$, 16-QAM case (45.5, 18.9 and 20, 11 percent respectively for 64-QAM). Increasing n_t to 8 from 4 roughly doubles the exact architectures' area which then becomes 2.3× larger for $n_t = 16$ (in the approximate case the average factor is 3.5×). Frequency-wise, the Sequential PAM SD's advantage is less than 4 percent at 16-QAM for $n_t = 8$ (i.e., 389 v. 377 MHz) and less than 8 percent at 64-QAM (345 v. 322 MHz), while it is diminishing for larger n_ts. Similarly, a-MultiSphere achieves 1.176 GHz ($|\mathcal{O}| = 16$) and 1 GHz ($|\mathcal{O}| = 64$) for $n_t = 8$ (the *FSD* respectively achieving 1.250 and 1.030 GHz). Fig. 13-right shows that setting $N_{PE} = |\mathcal{O}|$ and for 16-QAM modulation the sorting network accounts for up to 46 percent of the LLR processor's area ($n_t = 4$). Increasing n_t to 16, expands the sorting network's gate count by up to 2.14× due to the increase in the partial vector width and also increases the size of the LLR sub-processing arrays (Fig. 3), which account for up to 30.3 percent of the total LLR processor's gates (i.e., the flipped bit index processors). When $|\mathcal{O}| = N_{PE} = 64$, the sorting network dominates the total gate count (i.e., 63.5 up to 72.4 percent for $n_t = 16$ and $n_t = 4$ respectively). In all of the displayed sorting network cases and due to pipelining, N_{PE} does not affect the critical path as much as n_t does; thus the LLR processor achieves 1.25 GHz for $n_t = 4$ and up to 833 MHz for $n_t = 16$.

Multi-Engine Scalability and Detection Performance. For $N_{PE} \in \{8, 16, 32\}$ at 16-QAM, the average maximum (i.e., at 16 dB SNR) algorithmic speedup of exact detection based on the number of visited nodes is 4× up to 36× (Fig. 7-top). MultiSphere's VLSI post-synthesis processing throughput speedup for $N_{PE} = 32$ is close to the algorithmic speedup, i.e., approximately 29× against the Sequential PAM SD and 30× against a single MultiSphere processing element. Table 1 displays the maximum energy efficiency (E_{eff}) involving dynamic power consumption and the area required per exact and approximate PE at the maximum achievable frequency. Based on the above, we can configure the parallel engines for $N_{PE} \in \{8, 16, 32\}$ to respectively operate at 96.15, 20.45 and 10.68 MHz for which MultiSphere's dynamic power consumption decreases up to 27.4× (i.e., to 1.09, 0.26 and 0.11 mW per engine for $N_{PE} \in \{8, 16, 32\}$ respectively at 25° C and 0.9 V). Additional power consumption savings can be achieved through voltage scaling. Note that even though the exact SD in [43] has a lower area footprint due to a PSK-based enumeration, the proposed architectures achieve higher clock frequencies (Table 1).

In multi-carrier detection (Figs. 9, 10 and 14-left), MultiSphere is significantly more efficient than conventional parallelization via sequential SDs, and its efficiency increases with N_{PE}. When processing a frame with $N_{SC} = 52$, a single MultiSphere VLSI detector can achieve a speedup of 2.5× ($N_{PE} = 8$) to 15× ($N_{PE} = 32$) over 52 *Sequential PAM* detectors operating in parallel. This translates to a 7.4× higher energy efficiency ($N_{PE} = 8$, 10 dB SNR) which can increase up to 31.1× ($N_{PE} = 32$, 16 dB SNR). Moreover, MultiSphere features a notably smaller area footprint; at 752 ($N_{PE} = 8$) to 2,891 KGE ($N_{PE} = 32$), compared with 4,766 KGE for the *Sequential PAM*, 4,293 KGE the sequential PSK and 3,382 KGE the approximate sequential $l_{\widetilde{\infty}}$ SDs of [43]. Thus, MultiSphere's hardware efficiency is higher than all efficient sequential architectures: 6× ($N_{PE} = 8$, 10 dB SNR) up to 25× ($N_{PE} = 32$, 16 dB SNR) against our *Sequential PAM* and even 3× up to 13× against the very low complexity $l_{\widetilde{\infty}}$ SD of [43]. Figs. 14-left and 10 show that further increasing N_{SC}

5. Calculated as $\frac{\text{Throughput(bps)}}{\text{Area(GE)}}$ and $\frac{\text{dynamicPower(W)}}{\text{Throughput(bps)}}$. Throughput given by $\frac{n_t \cdot log_2(|\mathcal{O}|)}{\mathcal{L} \cdot t_{min}}$, t_{min} being the minimum clock period and $\mathcal{L}$ the clock cycles required for symbol detection ($\mathcal{L}$ equals to the average number of visited nodes in the exact and to $\frac{paths}{N_{PE}}$ in the approximate design).

6. Frequency scaling from $\mathcal{T}$ nm via multiplication by $\frac{\mathcal{T}}{45}$, power scaling from V_{DD} via multiplication by $(\frac{0.9}{V_{DD}})^2 \cdot \frac{45}{\mathcal{T}}$.

7. We finally note that input to the detection engines is assumed to be managed externally in line with the literature [8], [13], [18], [21], [28], [43], [44] and is beyond the scope of this work.

TABLE 1
MultiSphere's Per-engine Area, Power, Energy Efficiency and Speedup at Maximum Frequency Against the State-of-the-art

	Detector	$\mathcal{T}$ (nm)	N_{PE}	f (MHz)	Area[a] (KGE)	Power[a] (mW)[b]	E_{eff}[c] (pJ/bit)	Speedup
$\lvert\mathcal{O}\rvert=16,\ n_t=10$	Sequential PAM	45	1	385	91.66	3.8303	249.97	1.00[i]
	Sequential PSK [43]	250	1	332[d]	82.56[e]	N/A	N/A	0.86[i]
	MultiSphere	45	1	368	106.9	3.9403	268.66	0.95[i]
			8	333	751.7	27.416	415.56	3.47[i]
			16	322	1,473	54.967	234.12	15.8[i]
			32	307	2,891	99.747	256.07	28.7[i]
	FSD ($\lvert\mathcal{O}\rvert=64,\ n_t=16$)	45	1	893	1,175	401.28	299.62	1.00[j]
			4	833	3,171	880.73	176.14	3.73[j]
	a-MultiSphere ($\lvert\mathcal{O}\rvert=64,\ n_t=16$)		1	877	1,305	452.32	343.76	0.80[j]
			4	666	3,542	1,096.71	226.65	2.98[j]
	soft FSD ($\lvert\mathcal{O}\rvert=64,\ n_t=16$)	45	1	833	1,647	479.79	383.83	1.00[j]
			4	741	3,685	973.30	218.99	3.56[j]
	soft a-MultiSphere ($\lvert\mathcal{O}\rvert=64,\ n_t=16$)		1	666	1,742	423.50	423.50	0.80[j]
			4	666	4,059	1,024.67	256.71	3.20[j]
	Detector	**$\mathcal{T}$ (nm)**	**N_{PE}**	**f (MHz)**	**Area[a] (KGE)**	**Power[a] (mW)**	**H_{eff} (bps/GE)**	**E_{eff} (pJ/bit)**
	a-MultiSphere $\lvert\mathcal{O}\rvert=4,\ n_t=16$[g]	45	16	1,136	6,256	2,915.5	5,812.6	80.176
	[44] TASER[g]	40	1	404[d]	1,428	162.67[d]	226.19	503.62[d]
	a-MultiSphere $\lvert\mathcal{O}\rvert=64,\ n_t=4$[f,g]	45	12	1,111	653.6	240.79	40,800	9.0295
	[45] Complex K-Best, $K=10$[f,g]	130	1	1,205[d]	340.0	331.01[d]	8,497.1	114.57[d]
	a-MultiSphere $\lvert\mathcal{O}\rvert=64,\ n_t=8$[f,h]	45	24	1,111	3,823	1,519.2	13,950	27.531
	[46] Real K-Best [f,h]	90	1	364[d]	665.0	143.00[d]	13,154	16.346[d]

Listed architectures without a citation are synthesized from Verilog RTL via TSMC 45nm libraries, at 25°C and 0.9V. [a]*Hierarchical synthesis (multiple engines).* [b]*Averaged circuit activity via channel traces (SNR: 10-16 dB-$\lvert\mathcal{O}\rvert=16$, 17-23 dB-$\lvert\mathcal{O}\rvert=64$).* [c]*At 16 dB ($\lvert\mathcal{O}\rvert=16$) and 23 dB SNR ($\lvert\mathcal{O}\rvert=64$).* [d]*Scaled to 45 nm.* [e]*Scaled to 10×10 by 2.4.* [f]*l^1 norm.* [g]*16-bit datapath.* [h]*12-bit datapath.* [i]*Exact detection (footnote 5).* [j]*Approximate detection, 64 paths (footnote 5).*

also increases total latency in all cases. Still, for $N_{SC}=512$, a single MultiSphere detector with $N_{PE}=32$ at 16 dB SNR maintains approximately constant efficiency, at 280 pJ/bit and 124 bps/GE, while all sequential SDs are up to two orders of magnitude less efficient. Even the very low area footprint, approximate $l^{\sim}_{\infty}$ SD of [43], achieves only 1.02 bps/GE in this case.[8] Note that aforementioned results do not take into account the additional logic which would be required in order to distribute/collect the multiple subcarriers to/from the sequential SDs and which would procure an even more favourable result for MultiSphere. Fig. 14-right compares a-MultiSphere's and the FSD's energy and hardware efficiency at 500 MHz (well-below the frequency of Table 1 in order to allow performance projections for large N_{PE} values) for $N_{PE}\in\{1,\ldots,128\}$ in the case where the *FSD* expands two levels (i.e., 4,096 paths). Requiring just 512 paths to reach the same ML-approaching error rate (Fig. 11), a-MultiSphere achieves 8.72× higher energy efficiency and 9.63× higher hardware efficiency when taking into account the pipelined minimum search unit. The soft-output a-MultiSphere can have up to 7.04× higher area efficiency and 6.76× higher energy efficiency (for 4,096 v. 512 paths as in Fig. 12). A-MultiSphere's energy efficiency advantage can decidedly escalate when assuming a statically instantiated LLR processor capable to process the required number of paths in parallel (Fig. 14). Against the recent state-of-the-art, such as the 16×16 MIMO in [44] (only supporting up to QPSK), our 16-PE a-MultiSphere is an order of magnitude more hardware efficient and 6× more energy efficient. A 12-PE a-MultiSphere architecture attains a similar error-rate performance to the K-best SD in [45] ($K=10$), yet achieves 4× higher hardware and 12× higher energy efficiency. Finally, a 24-PE a-MultiSphere is of similar error-rate to the non-constant ($K\in[1,16]$) K-best SD in [46] and slightly more hardware but less energy efficient. We note though that we target a flexible proof-of-concept using complex enumeration, not a specifically optimized case; we also assume that the channel changes with every subcarrier while [46] does not detail power estimation assumptions.

8. The $l^{\sim}_{\infty}$ SD in [43] visits 5 percent fewer nodes than SE enumeration.

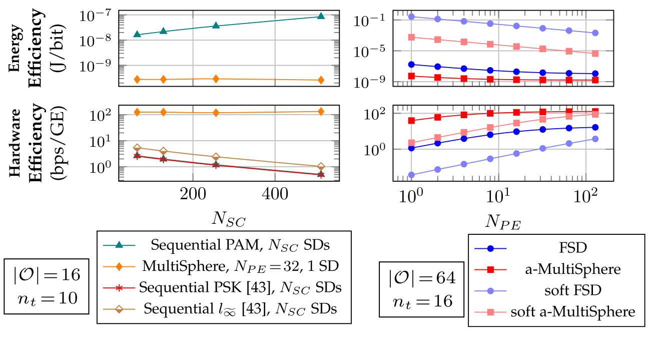

Fig. 14. MultiSphere (leftmost plots): energy and hardware efficiency when single detectors sequentially process multiple subcarriers ($N_{PE}=32$) against multiple sequential SDs one per subcarrier. a-MultiSphere (rightmost plots): energy and hardware efficiency v. N_{PE}. In this case, we extrapolate performance by instantiating the parallel framework at 2 ns, in order to develop a power and area model at 100 percent utilization.

TABLE 2
Isolated MCM and MT Module Complexity Per Detector, Type and MT Tree Size (TSMC 45nm, 25°C, 0.9V)

MCM: Complexity per detector			Complexity per MCM type[c]				
Detector	MCMs	$\|\mathcal{O}\|$	Figure of merit	[20][d] (neg)	[20][d] (2sC)	MCM-H	MCM-F
MultiSphere	$N_{PE}\cdot(4n_t+2\sqrt{\|\mathcal{O}\|}+6)$	16	ADP[e],[a]	186.39	218.75	252.90	230.86
			ADP[e],[b]	244.96	324.00	229.80	212.00
			EDP[f],[a]	0.039	0.049	0.055	0.055
			EDP[f],[b]	0.099	0.131	0.081	0.079
a-MultiSphere (hard/soft)	$N_{PE}\cdot(2n_t^2)$	64	ADP[e],[a]	309.04	488.37	279.48	264.6
			ADP[e],[b]	309.76	369.84	308.22	246.18
			EDP[f],[a]	0.062	0.092	0.062	0.063
			EDP[f],[b]	0.120	0.132	0.097	0.089

MT: Complexity per detector				**Node complexity per tree size[a],[i],[j],[k]**				
Detector	**MT Type**	**MT Processors**		**1**	**8**	**16**	**32**	**64**
MultiSphere	MT$_{eng}$	$N_{PE}\cdot[(\sqrt{\|\mathcal{O}\|}-1)n_t]+N_{PE}-1$	Area[g]	741	5428	11326	23121	46714
a-MultiSphere (hard/soft)		Paths$-1/\frac{N_{PE}}{4}\cdot\log_2 N_{PE}\cdot[\log_2 N_{PE}+1]$	Power[h]	0.1575	1.0503	2.1706	4.4022	8.8889
MultiSphere	MT$_{ni}$	$N_{PE}\cdot(\|\mathcal{O}\|+\sqrt{\|\mathcal{O}\|}-2)$	Area[g]	529	4317	9023	18408	37200
a-MultiSphere (hard/soft)		-/$n_t\cdot\log_2\|\mathcal{O}\|$	Power[h]	0.1086	0.8016	1.6555	3.3634	6.7961
MultiSphere	MT$_z$	$N_{PE}\cdot(\sqrt{\|\mathcal{O}\|}-1)$	Area[g]	765	5634	11768	24036	48572
a-MultiSphere (hard/soft)		-	Power[h]	0.1584	1.0540	2.1768	4.4166	8.9159

[a]*Hierarchical Synthesis.* [b]*Retiming Synthesis.* [c]*16-bit input.* [d]*Optimized for $|\mathcal{O}|=16$ or $|\mathcal{O}|=64$.* [e]*$GE\cdot ns$.* [f]*$mW\cdot ns^2$.* [g]*GE.* [h]*mW.*
[i]*16-bit key, 8-bit label.* [j]*Using i/o registers and 2ns period.* [k]*Total power assuming 100% utilization.*

Complexity Assessment-MCMs and MT Processors. To provide a clearer perspective to the reader, we conduct a complexity assessment of MultiSphere's main arithmetic modules i.e., the MCMs and the MT processors. Due to the folded design and its exact nature, MultiSphere requires fewer MCMs but more MT processors (Table 2). a-MultiSphere on the other hand features more computationally intensive yet simpler operations. We also assess the efficiency of the proposed MCMs against those in [20] via the area-delay and energy-delay products assuming 16-bit input for all cases. Note that [20] defines a single flexible MCM for $|\mathcal{O}| \in \{16, 64\}$. For fairness, we employ distinct, optimized versions per modulation. By "2sC" and "neg" we respectively distinguish between two's complement and negation units (Fig. 4). Results show that for hierarchy-preserving synthesis, [20] with two's complement is slightly more efficient at 16-QAM. At 64-QAM both of the proposed solutions are more efficient even against the simple negation of [20]. Notice that when the synthesis tool aggressively optimizes the design (retiming strategy), the proposed MCMs are more efficient in all cases. We chose the MCM-H due to the exploration scope of the paper, in line with hierarchy-preserving synthesis. Regarding the MT processors, MT_z is the most complex, but intentionally also the one least employed. Notice (Section 4) that the tree size inside the detectors has $O(\sqrt{|\mathcal{O}|})$ complexity, while the interconnection where MT_{eng} is employed has $O(N_{PE})$ complexity. Utilization of the MT processors in trees displays a close to linear behavior while the area and power of the tree are almost negligible compared to that of the rest of the PE (i.e., 3.6 and 3.1 percent of total respective area and power at 100 percent utilization for a tree of 64 MT_{eng} processors and $N_{PE}=1$ for a-MultiSphere). We note that the MT results for Table 2 employ i/o registers in every processor and thus more closely reflect the a-MultiSphere case. MultiSphere's MTs exhibit a very similar relative cost, though a single processor can have up to 79 percent lower area compared to Table 2. MultiSphere's critical path lies within the *TTP* and even a 512 MT_{eng} interconnection achieves below 3.26 ns delay. Moreover, the MT_{eng} tree attributes a small fraction to MultiSphere's dynamic power consumption i.e., 0.85 μW for $N_{PE}=32$ at 16 dB. Note that the *LLR processor* employs generic multipliers as the ones used for l^2 norm calculation; thus soft-a-MultiSphere retains the same MCM count. The additional processors are attributed to a) the sorting network (i.e., $\frac{1}{2}\cdot\log_2(N_{PE})\cdot[\log_2(N_{PE})+1]$ stages of $\frac{N_{PE}}{2}$, MT_{eng}-type processors per stage) and b) the LLR clipping processors ($n_t\cdot\log_2(|\mathcal{O}|)$, MT_{ni}-type processors).

6 Conclusions

This work proposes MultiSphere, the first method to consistently and massively parallelize large sphere decoders, and consequently the fundamental ML detection problem, in a nearly-"embarrassingly" parallel manner, while accounting for the transmission channel. Joint algorithmic/VLSI evaluation shows that MultiSphere is the first approach able to substantially and consistently reduce latency at a small complexity overhead. Our efficient VLSI architecture performs close to the algorithmic bound and in multi-carrier detection is up to two orders of magnitude more efficient than conventional parallelization employing the most efficient SDs in the literature. Moreover, a-MultiSphere's algorithmic performance enables our flexible VLSI framework to be up to an order of magnitude more efficient than highly optimized, state-of-the-art approaches. Besides large MIMO systems, MultiSphere enables the practical realization of a plethora of theoretical concepts the implementation of which is considered impractical. Such concepts include aggressive non-orthogonal multiple access (NOMA) schemes [47], [48], as

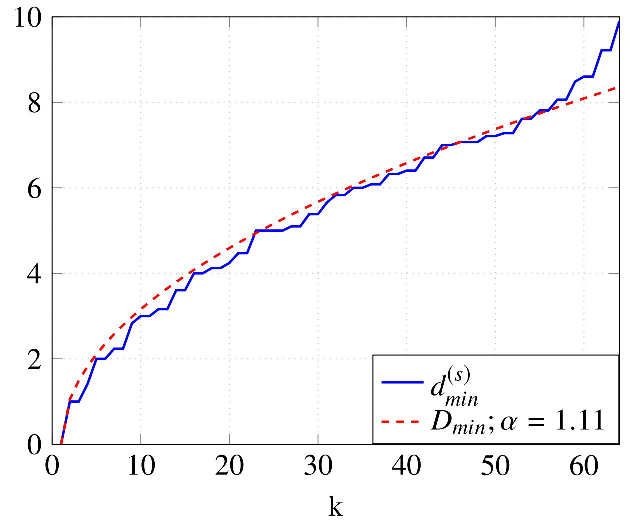

Fig. 15. D_{min} and $d_{min}^{(s)}(k)$ values for 64-QAM constellation with minimum distance of two between symbols.

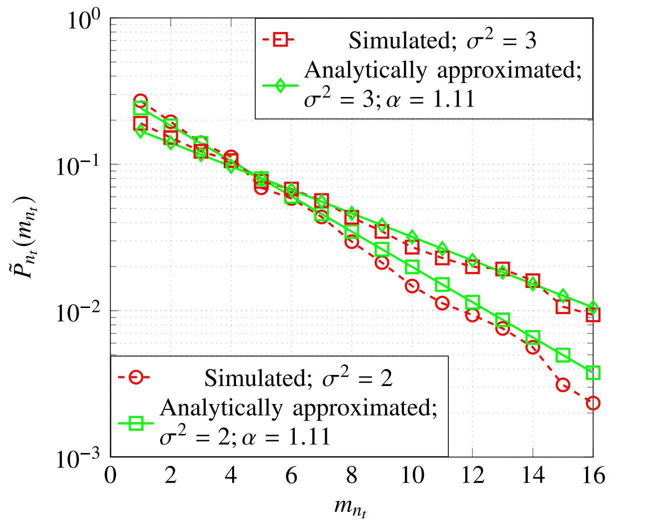

Fig. 16. Simulated and the analytically approximated $\tilde{P}_{n_t}$ as a function of m_{n_t}.

well as “Faster than Nyquist” transmissions, including the promising Spectrally Efficient Frequency Division Multiplexing (SE-FDM) scheme [49], [50].

APPENDIX
MULTISPHERE'S METRICS OF PROMISE (MOPS)

Here we first calculate an MoP that approximates the probability of an SD path to constitute the correct solution $\mathbf{s}^{(t)}$. Then, a simplified MoP is given that does not require the prior knowledge of the noise variance σ^2 but, as shown in Section 5, it is equally efficient with the original MoP in terms of its ability to reduce sphere decoding processing latency.

As described in Section 3.1, each tree path can be described by its relative position vector (RPV) $\mathbf{m}$, with elements m_l. Denoting as $\mathbf{x_m}$ the symbol vector related to the path $\mathbf{m}$, and with its lth element being x_{m_l}, our target is to find an MoP that approximates the probability $P\left[\mathbf{x_m} = \mathbf{s}^{(t)}\right]$. Using Bayes' chain rule, this probability can be expressed as

$$P\left[\mathbf{x_m} = \mathbf{s}^{(t)}\right] = \prod_{l=1}^{n_t} \tilde{P}_l[m_l], \tag{5}$$

with

$$\tilde{P}_l[m_l] = P\left[x_{m_l} = s_l^{(t)} \middle| \bigcap_{q=l+1}^{n_t} x_{m_q} = s_q^{(t)}\right], \tag{6}$$

and

$$\tilde{P}_{n_t}\left[m_{n_t}\right] = P\left[x_{m_{n_t}} = s_{n_t}^{(t)}\right]. \tag{7}$$

We first calculate the probability $\tilde{P}_{n_t}\left[m_{n_t}\right]$ for the highest SD tree level. This equals the probability that $s_{n_t}^{(t)}$ is the symbol with the m_{n_t}th smallest PD, or equivalently it is the m_{n_t}th closest QAM constellation symbol to the equivalent received observable $\overline{y}_{n_t}$ (see Eq. (3)),

$$\overline{y}_{n_t} = s_{n_t}^{(t)} + \overline{w}_{n_t}, \tag{8}$$

where $\overline{w}_{n_t}$ represents the equivalent additive white Gaussian noise of variance $\overline{\sigma}_{n_t}^2 = \sigma^2 / \left|R_{n_t n_t}\right|^2$. Calculating $\tilde{P}_{n_t}\left[m_{n_t}\right]$ is a non-trivial task that would require complicated integrations with no obvious closed-form solutions. In order to simplify this task, the corresponding probability is approximated by using the pre-calculated minimum distance values that have been used in Section 3.1.3. By $d_{min}^{(s)}(k)$ (with $k = 1, \ldots, |\mathcal{O}|$) we denote the kth sorted d_{min} value. Then, as shown in Fig. 15, we observe that the ordered $d_{min}^{(s)}(k)$ values can be well approximated by the function

$$d_{min}^{(s)}(k) \approx D_{min}(k) = \sqrt{\alpha(k-1)}, \tag{9}$$

with α depending on the minimum distance between the QAM constellation symbols. For a minimum constellation distance of two (where each QAM symbol dimension can take the values $\pm 1, \pm 3, \ldots$) a value of $\alpha = 1.11$ is chosen, which minimizes the mean-squared-difference between $d_{min}^{(s)}(k)$ and $D_{min}(k)$ for a 64-QAM constellation. Since, in contrast to $d_{min}^{(s)}(k)$, $D_{min}(k)$ is a strictly increasing function of the parameter k, we can approximate the probability $\tilde{P}_{n_t}\left[m_{n_t}\right]$ as

$$\tilde{P}_{n_t}\left[m_{n_t}\right] \approx P\left[D_{min}(m_{n_t}) \leq \left|\overline{w}_{n_t}\right| < D_{min}(m_{n_t}+1)\right]. \tag{10}$$

Since the norm of the noise is Rayleigh distributed, and by applying the approximation in (9), the above probability can be easily calculated as

$$\tilde{P}_{n_t}\left[m_{n_t}\right] \approx e^{-\frac{\alpha\left(m_{n_t}-1\right)\left|R_{n_t n_t}\right|^2}{2\sigma^2}} - e^{-\frac{\alpha m_{n_t}\left|R_{n_t n_t}\right|^2}{2\sigma^2}}. \tag{11}$$

Fig. 16 shows the simulated and the analytically approximated $\tilde{P}_{n_t}$ for an inner QAM constellation symbol and verifies the validity of the proposed approximation. After approximating $\tilde{P}_{n_t}\left[m_{n_t}\right]$ in (7), the probability $\tilde{P}_l[m_l]$ in (5) needs to be calculated to get $P\left[\mathbf{x_m} = \mathbf{s}^{(t)}\right]$. For all the SD tree layers l with $l < n_t$, the received observable after the QR decomposition is

$$\tilde{y}_l = \sum_{j=l+1}^{n_t} R_{lj} s_j^{(t)} + R_{ll} s_l^{(t)} + w_l. \tag{12}$$

Thus, under the assumption imposed by (5) that $x_{m_q} = s_q^{(t)}$ for all tree levels higher than l (i.e, that the corresponding symbols belong to the correct vector solution), the received observable at any level l, for any path, can be expressed as

$\overline{y}_l = s_l^{(t)} + \overline{w}_l$ similarly to (8), with $\overline{\sigma}_l^2 = \sigma^2/|R_{ll}|^2$. By using a similar reasoning with when calculating $\tilde{P}_{n_t}[m_{n_t}]$, $\tilde{P}_l[m_l]$ can be generalized as

$$\tilde{P}_l[m_l] \approx e^{-\frac{\alpha_l[m_l-1]|R_{ll}|^2}{2\sigma^2}} - e^{-\frac{\alpha_l m_l |R_{ll}|^2}{2\sigma^2}}, \tag{13}$$

for any $l = 1, \dots, n_t$, and with the total probability

$$\sum_{k=1}^{|\mathcal{O}|} \tilde{P}_l[m_l] \approx 1 - e^{-\frac{\alpha_l |\mathcal{O}| |R_{ll}|^2}{2\sigma^2}}, \tag{14}$$

asymptotically, for large $|\mathcal{O}|$, tending to the value of one as it should for the total probability. We note that the parameter α can, in general, differ with l, since different QAM modulation can be used per transmit antenna. Since the logarithmic function is monotonic, finding the most promising paths is equivalent to finding the paths for which the logarithm of their probability $P\left[\mathbf{x_m} = \mathbf{s}^{(t)}\right]$ is minimized. Therefore, an MoP for an SD tree path with RPV $\mathbf{m}$ can be given by

$$\mathcal{M}(\mathbf{m}) = -\sum_{l=1}^{n_t} \ln\left\{ e^{-\frac{\alpha_l[m_l-1]|R_{ll}|^2}{2\sigma^2}} - e^{-\frac{\alpha_l m_l |R_{ll}|^2}{2\sigma^2}} \right\}. \tag{15}$$

with

$$\mathcal{M}(\mathbf{m}) \approx -\ln\left\{ P\left[\mathbf{x_m} = \mathbf{s}^{(t)}\right] \right\}. \tag{16}$$

The MoP of (15) requires the prior knowledge of the σ^2 value. If, for any reason, this is not available, a simplified metric can be calculated instead. Since the exponential terms in (13) are exponentially decreasing with m_l, an upper bound of $\tilde{P}_l[m_l]$ can be calculated as

$$\tilde{P}_l[m_l] \leq e^{-\frac{\alpha_l[m_l-1]|R_{ll}|^2}{2\sigma^2}}, \tag{17}$$

and therefore, an approximate MoP can be defined as

$$\tilde{\mathcal{M}}(\mathbf{m}) = \frac{1}{2\sigma^2} \sum_{l=1}^{n_t} \alpha_l [m_l - 1] |R_{ll}|^2, \tag{18}$$

with

$$\tilde{\mathcal{M}}(\mathbf{m}) \leq -\ln\left\{ P\left[\mathbf{x_m} = \mathbf{s}^{(t)}\right] \right\}. \tag{19}$$

From (18) it can be easily seen that finding the paths with the smallest MoPs, does not really require the knowledge of σ^2, or even of the parameter α when the same QAM constellation is used from all transmit antennae. Therefore, the following simplified MoP can be used instead

$$\mathcal{M}_s(\mathbf{m}) = \sum_{l=1}^{n_t} \alpha_l [m_l - 1] |R_{ll}|^2. \tag{20}$$

Acknowledgments

The research leading to these results has been supported from the UKs Engineering and Physical Sciences Research Council (EPSRC Grant EP/M029441/1). The Authors would also like to thank the members of University of Surrey 5GIC (http://www.surrey.ac.uk/5GIC) for their support and C. Husmann for his assistance. Part of this work has been presented in IEEE GLOBECOM 2016 [1]. The work is related to the filed patent, Parallel Processing of Sphere Decoders and Other Vector Finding Approaches using Tree Search, UKIPO: 1509910.4, PCT/GB2016/051675.

References

[1] K. Nikitopoulos, D. Chatzipanagiotis, C. Jayawardena, and R. Tafazolli, "MultiSphere: Massively parallel tree search for large sphere decoders," in *Proc. IEEE Global Commun. Conf.*, 2016, pp. 1–6.

[2] Technical Specification Group Radio Access Network, "Study on scenarios and requirements for next generation access technologies," 3GPP, Tech. Rep. TR 38.913, Sep. 2016.

[3] "IEEE standard for information technology, part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, amendment 4: Enhancements for very high throughput for operation in bands below 6 ghz," IEEE Std 802.11ac-2013, Dec. 2013.

[4] "IEEE standard for information technology, part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE Std 802.11–2012, Mar. 2012.

[5] K. Nikitopoulos, J. Zhou, B. Congdon, and K. Jamieson, "Geosphere: Consistently turning MIMO capacity into throughput," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 631–642.

[6] Q. Yang, X. Li, H. Yao, J. Fang, K. Tan, W. Hu, J. Zhang, and Y. Zhang, "BigStation: Enabling scalable real-time signal processing in large MU-MIMO systems," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 399–410, 2013.

[7] E. Viterbo and J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1639–1642, Jul. 1999.

[8] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE J. Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1577, Jul. 2005.

[9] R. Courtland, "Transistors could stop shrinking in 2021, " *IEEE Spectrum*, vol. 53, no. 9, pp. 9–11, Sep. 2016.

[10] G. Fettweis, "5G–what will it be: The tactile internet," Keynote Presentation, in *Proc. IEEE Int. Conf. Commun.*, 2013.

[11] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[12] C. Hess, et al., "Reduced-complexity MIMO detector with close-to-ML error rate performance," in *Proc. ACM Great Lakes VLSI Symp.*, 2008, pp. 200–203.

[13] Z. Guo and P. Nilsson, "Algorithm and implementation of the k-best sphere decoding for mimo detection," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 3, pp. 491–503, Mar. 2006.

[14] H. W. Liang, W. H. Chung, H. Zhang, and S. Y. Kuo, "A parallel processing algorithm for Schnorr-Euchner sphere decoder," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Apr. 2012, pp. 613–617.

[15] M. S. Khairy, C. Mehlfhrer, and M. Rupp, "Boosting sphere decoding speed through graphic processing units," in *Proc. Eur. Wireless Conf.*, Apr. 2010, pp. 99–104.

[16] M. Wu, S. Gupta, Y. Sun, and J. R. Cavallaro, "A GPU implementation of a real-time MIMO detector," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2009, pp. 303–308.

[17] 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 Comput. Sci.*, vol. 18, pp. 2432–2435, 2013.

[18] C.-H. Yang and D. Marković, "A multi-core sphere decoder VLSI architecture for MIMO communications," in *Proc. IEEE Global Telecommun. Conf.*, 2008, pp. 1–6.

[19] C. H. Yang and D. Marković, "A 2.89mW 50GOPS 16x16 16-core MIMO sphere decoder in 90nm CMOS," in *Proc. Eur. Solid State Circuits Conf.*, Sep. 2009, pp. 344–347.

[20] C.-H. Yang and D. Markovic, "A flexible DSP architecture for MIMO sphere decoding," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 56, no. 10, pp. 2301–2314, Oct. 2009.

[21] M. Y. Huang and P. Y. Tsai, "Toward multi-gigabit wireless: Design of high-throughput MIMO detectors with hardware-efficient architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 2, pp. 613–624, Feb. 2014.

[22] L. G. Barbero and J. S. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," in *Proc. IEEE 7th Workshop Signal Process. Advances Wireless Commun.*, Jul. 2006, pp. 1–5.

[23] L. G. Barbero and J. S. Thompson, "Fixing the complexity of the sphere decoder for MIMO detection," *IEEE Trans. Wireless Commun.*, vol. 7, no. 6, pp. 2131–2142, Jun. 2008.

[24] J. Koo, S. Y. Kim, and J. Kim, "A parallel collaborative sphere decoder for a MIMO communication system," *J. Commun. Netw.*, vol. 16, no. 6, pp. 620–626, Dec. 2014.

[25] K. Nikitopoulos, A. Karachalios, and D. Reisis, "Exact max-log MAP soft-output sphere decoding via approximate Schnorr–Euchner enumeration," *IEEE Trans. Veh. Technol.*, vol. 64, no. 6, pp. 2749–2753, Jun. 2015.
[26] C. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Math. Program.*, vol. 66, no. 2, pp. 181–191, 1994.
[27] S. Chen, T. Zhang, and Y. Xin, "Relaxed-best MIMO signal detector design and VLSI implementation," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 15, no. 3, pp. 328–337, Mar. 2007.
[28] M. Shabany, K. Su, and P. Gulak, "A pipelined scalable high-throughput implementation of a near-ML K-best complex lattice decoder," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2008, pp. 3173–3176.
[29] C. Studer, A. Burg, and H. Bölcskei, "Soft-output sphere decoding: Algorithms and VLSI implementation," *IEEE J. Sel. Areas Commun.*, vol. 26, no. 2, pp. 290–300, Feb. 2008.
[30] B. Mennenga, A. von Borany, and G. Fettweis, "Complexity reduced soft-in soft-out sphere detection based on search tuples," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2009, pp. 1–6.
[31] E. P. Adeva and G. P. Fettweis, "Efficient architecture for soft-input soft-output sphere detection with perfect node enumeration," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 9, pp. 2932–2945, Sep. 2016.
[32] S. Hu and F. Rusek, "A soft-output MIMO detector with achievable information rate based partial marginalization," *IEEE Trans. Signal Process.*, vol. 65, no. 6, pp. 1622–1637, Mar. 2017.
[33] L. Zhuo, G. R. Morris, and V. K. Prasanna, "High-performance reduction circuits using deeply pipelined operators on FPGAs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 10, pp. 1377–1392, Oct. 2007.
[34] M. Huang and D. Andrews, "Modular design of fully pipelined reduction circuits on FPGAs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 9, pp. 1818–1826, Sep. 2013.
[35] L. G. Barbero and J. S. Thompson, "Rapid prototyping of a fixed-throughput sphere decoder for MIMO systems," in *Proc. IEEE Int. Conf. Commun.*, 2006, pp. 3082–3087.
[36] C. Studer and H. Bölcskei, "Soft-input soft-output sphere decoding," in *Proc. IEEE Int. Symp. Inf. Theory*, 2008, pp. 2007–2011.
[37] A. Burg, M. Wenk, and W. Fichtner, "VLSI implementation of pipelined sphere decoding with early termination," in *Proc. IEEE Eur. Signal Process. Conf.*, 2006, pp. 1–5.
[38] M. Zuluaga, P. Milder, and M. Püschel, "Streaming sorting networks," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 4, pp. 55:1–55:30, May 2016.
[39] D. Wübben, R. Böhnke, J. Rinas, V. Kühn, and K.-D. Kammeyer, "Efficient algorithm for decoding layered space-time codes," *IEEE Electron. Lett.*, vol. 37, no. 22, pp. 1348–1350, Oct. 2001.
[40] Rice Univ. Wireless Open Access Research Platform (WARP), (2006). [Online]. Available: http://warp.rice.edu/trac
[41] M. Shabany and P. Gulak, "Scalable vlsi architecture for k-best lattice decoders," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2008, pp. 940–943.
[42] L. G. Barbero and J. S. Thompson, "Extending a fixed-complexity sphere decoder to obtain likelihood information for turbo-MIMO systems," *IEEE Trans. Veh. Technol.*, vol. 57, no. 5, pp. 2804–2814, Sep. 2008.
[43] C. Studer, M. Wenk, and A. Burg, *VLSI Implementation of Hard- and Soft-Output Sphere Decoding for Wide-Band MIMO Systems*. Berlin, Germany: Springer, 2012.
[44] O. C. neda, T. Goldstein, and C. Studer, "Data detection in large multi-antenna wireless systems via approximate semidefinite relaxation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 12, pp. 2334–2346, Dec. 2016.
[45] M. Mahdavi and M. Shabany, "Novel MIMO detection algorithm for high-order constellations in the complex domain," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 5, pp. 834–847, May 2013.
[46] M. Y. Huang and P. Y. Tsai, "Toward multi-gigabit wireless: Design of high-throughput MIMO detectors with hardware-efficient architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 2, pp. 613–624, Feb. 2014.
[47] M. Vameghestahbanati, E. Bedeer, I. Marsland, R. H. Gohary, and H. Yanikomeroglu, "Enabling sphere decoding for SCMA," *IEEE Commun. Lett.*, vol. 21, no. 12, pp. 2750–2753, Dec. 2017.
[48] R. Hoshyar, R. Razavi, and M. Al-Imari, "LDS-OFDM an efficient multiple access technique," in *Proc. IEEE 71st Veh. Technol. Conf.*, May 2010, pp. 1–5.
[49] T. Xu and I. Darwazeh, "M-QAM signal detection for a non-orthogonal system using an improved fixed sphere decoder," in *Proc. 9th Int. Symp. Commun. Syst. Netw. Digital Sign*, Jul. 2014, pp. 623–627.
[50] J. Mazo, "Faster-than-nyquist signaling," *The Bell Syst. Tech. J.*, vol. 54, no. 8, pp. 1451–1462, 1975.

Konstantinos Nikitopoulos (M'07) is an associate professor with the Institute for Communication Systems, Electrical and Electronic Engineering Department, University of Surrey, Guildford, United Kingdom. He is a member of the 5G Innovation Centre, where he is leading the Proof-of-Concept and mm-Wave Solutions work area. Before joining the University of Surrey, he has held research positions with RWTH Aachen University, with the University of California at Irvine, Irvine, and with the University College London. He has also been a consultant for the Hellenic General Secretariat for Research and Technology, where he also served as a National Delegate of Greece to the Joint Board on Communication Satellite Programs of European Space Agency. He is a recipient of the prestigious First Grant of the UKs Engineering and Physical Sciences Research Council. He is a member of the IEEE.

Georgios Georgis received the BSc degree in physics from the Aristotle University of Thessaloniki, and the MSc and PhD degrees in computing from the University of Athens, Greece. His research interests include the design of parallel algorithms and architectures for real-time multi-dimensional signal processing, artificial intelligence and expert systems. He is currently a research fellow in the 5G Innovation Centre, University of Surrey in Guildford, United Kingdom. He is a member of the IEEE.

Chathura Jayawardena received the BEng degree in electronic engineering and the MSc degree in mobile communications from the University of Surrey, Guildford, United Kingdom, in 2014 and 2015, respectively. He is currently working toward the PhD degree in electronic engineering at Institute for Communication Systems, University of Surrey. His research interests include signal processing for communications, with an emphasis on detection methods for non-orthogonal transmission schemes. He is a student member of the IEEE.

Daniil Chatzipanagiotis received the BEng degree in electronic engineering and the MSc degree in mobile communications from the University of Surrey, Guildford, United Kingdom, in 2015 and 2016, respectively. His research interests include signal processing for communications, with an emphasis on detection methods for non-orthogonal transmission schemes.

Rahim Tafazolli is the professor of Mobile and Satellite Communications since April 2000, director of ICS since January 2010 and the founder and director of 5G Innovation Centre, University of Surrey, United Kingdom. He has more than 25 years of experience in digital communications research and teaching. He has authored and co-authored more than 500 research publications. He is regularly invited to deliver keynote talks and distinguished lectures to International conferences and workshops. He is co-inventor on more than 30 granted patents, all in the field of digital communications. He is regularly invited by many governments for advice on 5G technologies. He was advisor to the Mayor of London in regard to the London Infrastructure Investment 2050 Plan during May and June 2014. He has given many interviews to International media in the form of television, radio interviews and articles in international press. In 2011, he was appointed as fellow of Wireless World Research Forum (WWRF) in recognition of his personal contributions to the wireless world as well as heading one of Europes leading research groups. He is a senior member of the IEEE.