

# PARALLEL HIGH THROUGHPUT SOFT-OUTPUT SPHERE DECODER

Q. Qi, C. Chakrabarti

School of Electrical, Computer and Energy Engineering  
Arizona State University, Tempe, AZ 85287-5706  
{qi,chaitali}@asu.edu

## ABSTRACT

Multiple-Input-Multiple-Output communication systems demand fast sphere decoding with high performance. We propose a high throughput soft-output fixed complexity sphere decoder (PFSD) that is parallel and has comparable performance to list fixed complexity sphere decoder (LFSD) and  $K$ -best sphere decoder. In addition, we propose a parallel QR decomposition algorithm to lower the preprocessing overhead, and a low complexity LLR algorithm to allow parallel update of LLR values. We demonstrate the BER and computation complexity advantages of the PFSD algorithm in a  $4 \times 4$  16-QAM system. The PFSD algorithm has been mapped onto Xilinx XC4VLX160 FPGA. The resulting PFSD decoder can produce 8 candidate vectors per clock cycle, and achieve upto 75Mbps throughput for  $4 \times 4$  64-QAM configuration at 100MHz with low control overhead.

**Index Terms**— Sphere Decoding, Parallel Implementation, FPGA

## 1. INTRODUCTION

The increasing demand of robust and high throughput mobile systems has spear-headed the development of multiple-input multiple-output (MIMO) communication systems. The performance gain of a MIMO system comes at the cost of increasing design complexity. One of the most important modules in a MIMO system is the signal detector. Maximum-likelihood (ML) based signal detectors are impractical for high data rate MIMO systems, since their complexity increases exponentially with signal dimension. Active research on low-complexity and near ML MIMO detectors have generated several solutions, including zero-forcing equalization (ZF) [1], nulling and canceling (NC) [1] and sphere decoding (SD) [2]. Of these approaches, the SD algorithm is the most promising. It offers low complexity and good bit-error-rate (BER) performance under a variety of Signal-to-Noise (SNR) and constellation conditions.

A soft-output sphere decoder typically consists of a list generator that finds a set of candidate symbol vector, and a

log-likelihood (LLR) generator that calculates the soft-output bit value for the MIMO channel decoder. VLSI implementation of SD detectors, such as [3], focus on reducing the computation complexity of only the list generator. Recently, a high speed systolic-like soft-output sphere decoder has been proposed in [4]. However, the LLR generator involves sorting large number of candidate solutions, which incurs a high implementation complexity, and limits the throughput of the MIMO detector. In this paper, we present a new algorithm and architecture for a soft-output fixed complexity sphere decoder. The main contributions of this paper are listed below.

1. We introduce a *parallel fixed complexity sphere decoding* (PFSD) algorithm, and investigate its performance under different SNR and parallelization parameters. We found that for a  $4 \times 4$  16-QAM system, the PFSD provides comparable BER performance with the list fixed complexity decoder (LFSD) [5].
2. We introduce a *parallel QR decomposition* algorithm, which complements the PFSD by sharing intermediate results from multiple QR decompositions. Its computation complexity asymptotically approaches 3 times that of a single QR decomposition regardless of the input matrix size.
3. We introduce a *low complexity LLR* algorithm for PFSD that allows parallel update of LLR values. It has 85.7% less compare operations than a full list search based LLR algorithm.
4. We map PFSD algorithm onto XILINX XC4VLX160 FPGA, which can deliver up to 75Mbps throughput for  $4 \times 4$  systems with 64-QAM configuration soft-decision decoder at 100MHz with minimum control overhead.

The rest of the paper is organized as follows. We briefly review the sphere decoding algorithm in Section 2. Section 3 presents the PFSD, the parallel QR decomposition and the low complexity LLR algorithms. Section 4 provides algorithm simulation results. Section 5 discusses the hardware architecture for PFSD. The conclusion is given in Section 6.

## 2. PRELIMINARIES

A MIMO system with spatial multiplexing signaling consists of  $M_T$  transmit and  $M_R$  receive antennas. Let  $\underline{y}$  be the  $M_R \times 1$  vector of received symbols, given by

---

This work was supported in part by a grant from NSF CSR-0615135.

$$\underline{y} = H\underline{s} + \underline{n} \quad (1)$$

where  $H$  is an  $M_R \times M_T$  complex channel matrix,  $\underline{s} = [\tilde{s}_1, \tilde{s}_2, \dots, \tilde{s}_{M_T}]^T$  is an  $M_T \times 1$  vector of transmitted symbols, and  $\underline{n}$  is an  $M_R \times 1$  noise vector.

A MIMO detector generates a set of symbol candidates  $\{\underline{s}\}$  according to the following function

$$\|\underline{y} - H\underline{s}\|^2, \underline{s} \in \mathbb{O} \quad (2)$$

where  $\mathbb{O}$  denotes a set of all  $M^{M_T}$  transmitted symbol vectors. A set of possible transmitted coded bit vectors  $\{\underline{x}\}$  is obtained by demodulating symbol set  $\{\underline{s}\}$ . The channel decoder then uses the a posteriori log-likelihood ratio value (LLR) of the bits from de-interleaved  $\{\underline{x}\}$  to calculate the likely transmitted data sequence  $\hat{\underline{u}}$ . The  $k$ th bit LLR is calculated by searching minimum norms in  $\{\underline{x}\}$ . The low complexity *Max-log approximation* LLR  $L(x_k|\underline{y})$  is calculated as

$$L(x_k|\underline{y}) \approx \min_{\underline{x} \in \mathbb{X}_{k,+1}} \frac{\|\underline{y} - H\underline{x}\|^2}{\sigma^2} - \min_{\underline{x} \in \mathbb{X}_{k,-1}} \frac{\|\underline{y} - H\underline{x}\|^2}{\sigma^2} \quad (3)$$

where set  $\mathbb{X}_{k,+1}$  and  $\mathbb{X}_{k,-1}$  denote subsets of  $\{\underline{x}\}$  with the  $k$ th bit  $x_k = +1$  and  $x_k = -1$ , respectively. The  $k$ th bit LLR of  $\underline{x}$  requires two bit vectors with opposite binary values in the  $k$ th bit position. One of the bit vectors is always the ML solution. The other bit vector is the complementary solution, whose  $k$ th bit is defined as the *counter-hypothesis* bit.

We assume that the system is full rank, where  $M_R \geq M_T$ . Also, the channel matrix  $H$  and the noise variance  $\sigma^2$  are known to the receiver. To avoid exhaustive search, approximate LLRs are calculated by using soft sphere decoding (SD) algorithm, which finds  $K$  norm solutions to equation (2) within a hypersphere bounded by radius  $r$ , where  $K \ll M^{M_T}$ .

A SD algorithm converts the original least square problem to a recursive tree search problem by transforming equation (2) to the following form through  $QR$  decomposition.

$$\mathcal{E} = \|\hat{\underline{y}} - R\underline{s}\|^2 \quad (4)$$

$$= \sum_{i=M_T}^1 \left| \hat{y}_i - \sum_{j=M_T}^{i+1} r_{ij}s_j \right|^2 - r_{ii}s_i^2 \quad (5)$$

where  $H = QR$  and  $\hat{\underline{y}} = Q^H \underline{y} = R\underline{s}^{ZF}$ .  $Q$  is an  $M_R \times M_R$  orthogonal matrix, and  $R$  is an  $M_R \times M_T$  upper triangular matrix.  $Q^H$  is the complex conjugate transpose of  $Q$ ,  $\underline{s}^{ZF} = R^{-1}Q^H \underline{y}$  is the zero-forcing (ZF) solution, and  $\mathcal{E}$  is the squared distance between estimated and transmitted signals.

Equation (5) shows that a SD algorithm searches candidate solutions in an inverse  $M_T + 1$  level  $M$ -ary tree, where each node has  $M$  child nodes except the leaf nodes on level 1. For description convenience, we define  $\underline{s}_{(i)} = [s_i, s_{i+1}, \dots, s_{M_T}]^T$  as the *partial vector symbol* candidate,  $\mathcal{V}_{i+1} = \hat{y}_i - \sum_{j=M_T}^{i+1} r_{ij}s_j$  as the corresponding *residual vector metric* and  $\mathcal{D}_i(\underline{s}_{(i)}) = \mathcal{V}_{i+1}(\underline{s}_{(i+1)}) - r_{ii}s_i$  as the respective *branch metric*. Each branch is associated with a branch metric. The *partial Euclidean distance metric* (PED),  $\mathcal{T}_i = \sum_{j=M_T}^i |\mathcal{D}_j(\underline{s}_{(j)})|^2$ , defines the accumulative branch metrics along a path from the root node to a node at level  $i$ .

Existing SD algorithms are based on either the *depth-first* or *breadth-first* search. The depth-first SD algorithms [6] generate one candidate solution at a time, and reduce  $r$  based on the current best solution. The breadth-first SD algorithm [7], also known as  $K$ -best algorithm, generates  $KM$  partial vector symbols at each level and keeps the  $K$  smallest PED paths. From hardware implementation stand-point, depth-first SD algorithms are recursive and difficult to parallelize, and  $K$ -best algorithm requires large number of sorting operations for moderate  $K$  and  $M$ . Both issues are circumvented by fixed complexity SD algorithms [8] [9].

Fixed complexity SD algorithms are breadth-first SD algorithms that generate a set of transmitted vectors by traversing fixed paths from the root level to the leaf level. The entire tree search procedure is defined by the cardinality vector  $\underline{t} = [t_1, t_2, \dots, t_{M_T}]$ . At level  $i$ , each parent node enumerates  $t_i$  child nodes in increasing order of  $\mathcal{D}_i(\underline{s}_{(i)})$ . All expanded nodes are kept and sorting is restricted to sibling nodes. The cardinality vector  $\underline{t}$  and symbol detection order can greatly impact the performance of a fixed complexity SD algorithm. The algorithm in [8] offers an effective solution, and it is denoted as FSD algorithm in the following sections.

The FSD algorithm only has two types of node expansions, full expansion (FS) and single expansion (SS). In FS, a parent node expands and keeps all  $M$  child nodes. In SS, only the child node with the smallest  $\mathcal{D}_i(\underline{s}_{(i)})$  is kept. The top  $p$  levels of solution set generation are FS, and the remaining levels are SS. The symbols in FS levels are detected in ascending order of their post-detection noise amplification, and the symbols in SS levels are detected in descending order of their post-detection noise amplification. Next we describe a high throughput soft-output parallel fixed complexity SD (PFSD) algorithm based on [8].

### 3. PARALLEL FIXED COMPLEXITY SPHERE DECODING

Existing soft decision fixed complexity sphere decoding algorithms [5] [10] provide high diversity in the bit values by either selective child node enumeration, or expansion of larger number of child nodes at successive levels. Both procedures introduce data dependency between sibling nodes, and require additional computation. Our PFSD algorithm eliminates this dependency by producing soft decision outputs from multiple hard decision FSDs. Furthermore, PFSD provides diverse bit values, and improves the BER performance when compared to the LFSD algorithm with similar complexity.

The proposed PFSD algorithm is presented in Alg. 1. It starts with the same channel matrix ordering step as the FSD algorithm [8], which produces a  $1 \times M_T$  permutation vector  $\underline{k}_1$ . Additional  $M_T - 1$  permutation vectors are derived from  $\underline{k}_1$  iteratively (lines 2-5). Vector  $\underline{k}_i$  must guarantee that its last element  $\underline{k}_{i,M_T}$  differs from  $\underline{k}_{j,M_T}$  for all  $i \neq j$ . A new  $H_i$  is obtained by column-wise permuting the original  $H$  according to  $\underline{k}_i$ . In

---

**Algorithm 1** Soft decision PFSD algorithm
 

---

**Require:**  $M, M_T, \underline{y}, H$ 

- 1: {Channel Matrix Ordering:}
  - 2: Produce permutation vector  $\underline{k}_1 = [k_{1,1}, k_{1,2}, \dots, k_{1,M_T}]$
  - 3: **for**  $i = 2$  to  $M_T$  **do**
  - 4:   Construct  $\underline{k}_i$ , where  $k_{i,M_T} \neq k_{j,M_T}, \forall j < i$
  - 5: **end for**
  - 6: {Solution Set Generation:}
  - 7: Set  $\mathbb{S} = \emptyset$
  - 8: **for**  $i = 1$  to  $M_T$  **do**
  - 9:   Generate  $H_i$  by permuting  $H$  column-wise according to  $\underline{k}_i$
  - 10:    $[Q_i, R_i] = qr(H_i), \hat{\underline{y}}_i = Q_i^H \underline{y}$
  - 11:    $\underline{t}_i = [1, \dots, 1, \hat{M}], \hat{M} \leq M$
  - 12:   {FSD tree search:}
  - 13:   Input:  $R_i, \hat{\underline{y}}_i, \underline{t}_i$
  - 14:   Output: Solution subset  $\mathbb{S}_i$
  - 15:   Inverse permute candidate vectors in  $\mathbb{S}_i$  using  $\underline{k}_i$
  - 16:    $\mathbb{S} = \mathbb{S}_i \cup \mathbb{S}$
  - 17: **end for**
  - 18: {Calculate Outputs:}
  - 19: Assign minimum weight vector in  $\mathbb{S}$  to  $\underline{s}^{ML}$
  - 20: Calculate LLR values  $\underline{l}$  from  $\mathbb{S}$
  - 21: **return** hard and soft decision outputs
- 

1 2 3 4 5 6 7 8								1 2 3 4 5 6 7 8									
$k_1$	a	b	c	d	e	f	g	h	$k_1$	a	b	c	d	e	f	g	h
$k_2$	a	b	c	d	e	f	h	g	$k_2$	h	b	c	d	e	f	g	a
$k_3$	a	b	c	d	h	g	f	e	$k_3$	h	a	c	d	e	f	g	b
$k_4$	a	b	c	d	h	g	e	f	$k_4$	h	a	b	d	e	f	g	c
$k_5$	h	e	f	g	a	b	d	c	$k_5$	h	a	b	c	e	f	g	d
$k_6$	h	e	f	g	a	b	c	d	$k_6$	h	a	b	c	d	f	g	e
$k_7$	h	e	f	g	c	d	b	a	$k_7$	h	a	b	c	d	e	g	f
$k_8$	h	e	f	g	c	d	a	b	$k_8$	h	a	b	c	d	e	f	g

(a)

(b)

**Fig. 1:** Permutation Order Comparison of: (a) Parallel QR and (b) Serial QR Decompositions for  $8 \times 8$  Channel Matrix.

the solution set generation step, PFSD produces candidate set  $\mathbb{S}_i$  for each pair of  $H_i$  and  $\underline{y}$  by performing FSD search with cardinality vector  $\underline{t}_i = [1, 1, \dots, \hat{M}]$  (lines 9-16). Hence, there are  $M_T$  candidate sets and each set has  $\hat{M}$  vectors. Note that when  $\underline{t}_{i,M_T} = \hat{M} = M$ , the first decoding level is fully expanded. It can be relaxed to a partial expansion level ( $\underline{t}_{i,M_T} < M$ ) to reduce overall computation complexity. However, this results in performance degradation as demonstrated in the simulation results in Section 4. Finally, PFSD finds the quasi-ML solution  $\underline{s}^{ML}$  and calculates the  $1 \times M_c M_T$  LLR vector  $\underline{l}$  from  $M_T \hat{M}$  vector candidates (lines 19-20).

Essentially, PFSD produces soft decision outputs by performing multiple hard decision FSDs. The symbol detection order of the  $i$ th FSD is uniquely determined by its permutation vector  $\underline{k}_i$ . Element  $k_{i,M_T}$  specifies of which symbol in  $\underline{s}$  to detect first. Since  $k_{i,M_T}$  differs from  $k_{j,M_T}$  for  $i \neq j$ , each symbol element of vector  $\underline{s}$  becomes the 1st detected symbol exactly

once. Furthermore, the  $i$ th FSD considers  $\hat{M}$  symbol values for its 1st detected symbol  $s_{i,M_T}$ . For sufficiently large  $\hat{M}$ , bit 1 and 0 exist at every bit position of  $s_{i,M_T}$ . Hence, PFSD guarantees the existence of counter-hypothesis bit (defined in section 2). This improves the reliability of LLRs.

### 3.1. Parallel QR Decomposition for PFSD

To support multiple tree searches in PFSD, a QR decomposition is needed for each FSD, which adds computation complexity to the overall algorithm. We propose a parallel QR decomposition algorithm that lowers computation complexity by sharing the intermediate results and combining the channel matrix ordering step with QR decomposition. Since QR decomposition is an iterative process starting from the left most column of  $H_i$ , matrices  $H_i$  and  $H_j$  differing only in the two right most columns share most intermediate QR decomposition results. The difference between  $H_i$  and  $H_j$  is fully described by their respective permutation vector  $\underline{k}_i$  and  $\underline{k}_j$ .

Our parallel QR decomposition algorithm generates a set of permutation vectors that maximize computation sharing. Figure 1 illustrates different permutation vectors resulting from parallel and serial QR decomposition. For serial QR, each decomposition is calculated separately. Alphabets  $a$  through  $h$  are used to denote the values of permutation vector  $\underline{k}_j$ ,  $1 < j < 8$ . For parallel QR, same color filling is applied to a block of vector elements to indicate identical intermediate decomposition results.

Serial QR is similar to the Layered Orthogonal Decoding (LORD) in [11]. However, both parallel and serial QR algorithm includes symbol detection ordering where  $\underline{k}_1$  is generated according to FSD channel ordering rule [8]. For a  $8 \times 8$  matrix, the parallel QR requires 34.1% and 35.2% less multiplication and addition operations than the serial QR. As matrix rank increases, the computation complexity of parallel QR approaches 3 times that of a single QR decomposition.

### 3.2. Low Complexity LLR Algorithm for PFSD

The low complexity LLR algorithm for PFSD produces sub-optimal soft-output values for a given set of candidate vectors in order to reduce number of sorting and demodulation operations. It searches all candidate vectors for the quasi-ML solution, and only a subset of candidate vectors for each counter-hypothesis bit solution. This reduces the computation complexity of LLR generation significantly.

The low complexity LLR algorithm starts by receiving  $M_T$  subsets of candidate symbol vectors ( $\mathbb{S}_i$ ) and their path metrics ( $\mathbb{E}_i$ ) from the PFSD solution generation step. The algorithm searches and saves minimum PED values  $\mathcal{E}_{min}^{x_{i,j,b}^0}$  and  $\mathcal{E}_{min}^{x_{i,j,b}^1}$  within  $\mathbb{E}_i$ . The  $b$ th bit of the  $j$ th bit vector is denoted by  $x_{i,j,b}$ , where  $1 \leq b \leq M_c$  and  $x_{i,j,b} \in \mathbb{X}_i$ ;  $x_{i,j,b}^0$  and  $x_{i,j,b}^1$  represents  $x_{i,j,b}$  with bit value 0 and 1 respectively, and  $\mathcal{E}^{x_{i,j,b}^0}$

**Table 1:** SD algorithm operation count comparison

Real value operations	Addition	Multiplication
PFSD (64:4x[1,1,1,16])	3456	1680
PFSD (32:4x[1,1,1,8])	1728	848
LFSD (64:[1,2,2,16])	2080	1008
$K$ -best ( $K=16$ )	3024	1872

and  $\mathcal{E}^{x_{i,j,b}^1}$  denote their PED values. Variable  $x_{i,j,b}$  is obtained by demodulating  $s_{i,j,M_T}$ . Variable  $s_{i,j,M_T}$  is the 1st detected symbol in the  $j$ th candidate symbol vector  $\underline{s}_{i,j}$ , where  $\underline{s}_{i,j} \in \mathbb{S}_i$ .

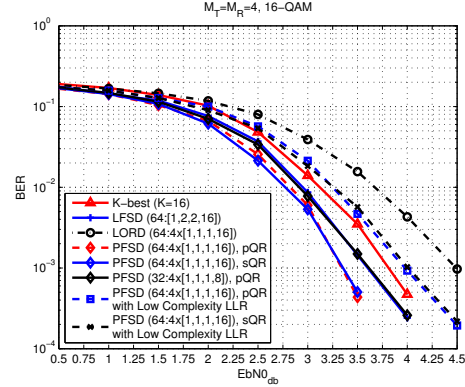
Since path metrics  $\mathcal{E}_{min}^{x_{i,j,b}^0}$  and  $\mathcal{E}_{min}^{x_{i,j,b}^1}$  only depend on subset  $\mathbb{S}_i$  and  $\mathbb{E}_i$ ,  $\hat{M}(M_T - 1)$  candidate vectors from other subsets are ignored. However, the quasi-ML solution is selected from all  $\hat{M}M_T$  vectors. This results in suboptimal search of the counter-hypothesis bits, and full search of the quasi-ML bits.

Since  $\mathbb{S}_i$  and  $\mathbb{S}_j$  are generated from different FSDs, this allows parallel LLR value computation without compromising the quasi-ML solution. For a  $4 \times 4$  MIMO system, the low complexity LLR algorithm has 85.7% and 75% less compare and demodulation operations than the full LLR search algorithm where all  $\hat{M}(M_T - 1)$  symbol candidate vectors are used to find each counter-hypothesis solution.

#### 4. SIMULATION RESULTS

In this section, we compare the Monte-Carlo simulation results of PFSD, LFSD and  $K$ -best sphere decoding algorithms for a  $4 \times 4$  MIMO system. Rate  $R_c = 1/2$  Turbo encoder is used to encode 4096 bits long source data. Interleaved encoder outputs are mapped onto gray code 16-QAM constellations. Block Rayleigh fading is used to model the channel, and the channel matrix  $H$  stays constant for every 16 consecutively transmitted symbol vectors. The fading coefficients  $h_{ij}$  are independent complex unit variance Gaussian variables. The energy per transmitted information bit is defined as  $E_b = 1$ . The MIMO detector calculates the soft-value information using the Max-log approximation for each received symbol vector. Each LLR is a signed 10-bit long number with 6 bits in the fractional part. The LLR clipping values of  $\pm 8$  are used for a prescribed bit when no counter-hypothesis bit is found in the candidate solution set [12]. The Turbo decoder takes the soft outputs from the MIMO detector, and decodes the information bits after eight iterations.

Figure 2 shows the BER performance of LFSD, PFSD and  $K$ -best decoders. The LFSD algorithm is a soft-output extension of the FSD algorithm [5] with cardinality vector [1,2,2,16]. There are three setups for the PFSD decoder, which differ in candidate number, QR ordering and LLR calculation. The QR ordering schemes for PFSDs are illustrated in Figure 1. The  $K$ -best decoder finds the best 16 nodes out of 256 candidate nodes at each decoding level.



**Fig. 2:** BER performance of the LFSD, the PFSD and the  $K$ -best algorithms with  $R_c = 1/2$  Turbo code and 16-QAM.

The 64 candidate PFSD outperforms the LFSD by 0.28dB, and the  $K$ -best by nearly 0.5dB. Even the 32 candidate PFSDs provide comparable results to 64 candidate LFSD with half the number of candidate vectors. The performance advantage of 64 and 32 candidate PFSD decoders is due to the existence of counter-hypothesis bits for all bit positions in their candidate solution sets. These generate more reliable LLR values than the LFSD and the  $K$ -best decoders. The 64 candidate PFSD with parallel QR and serial QR have near identical results; the performance difference is less than 0.1dB at  $10^{-3}$  BER. This is due to multiple QR decompositions, where no single sub-optimal channel matrix ordering adversely affects overall decoder performance.

In low complexity LLR, the counter-hypothesis bit path metric is calculated from  $\frac{1}{M_T}$  candidate list. This provides significant sorting saving at the expense of small BER degradation. Hence, when low complexity LLR algorithm is used for 64 candidate PFSD, the BER performance degrades by 0.7dB, and the sorting complexity is reduced by 75%. The results also show that channel matrix ordering has no significant impact on PFSD with low complexity LLR.

Table 1 shows the tree search complexity of all four decoders.  $l^2$ -norm is used for calculating  $|\mathcal{D}_i(s_{(j)})|^2$ . We assume each complex multiplication requires 5 additions and 3 real multiplications. The 64 candidate PFSD has the best performance but is computationally more complex compared to the 32 candidate PFSD and LFSD. The 32 candidate PFSD has 17% less multiplications and additions than the LFSD. In addition, the PFSD algorithm can also trade off computation complexity with BER by varying the number of candidate nodes expanded at the top level of each FSD.

An additional feature of the PFSD algorithm is its lower overall computation time. Recall that the PFSD algorithm is inherently parallel, where  $M_T$  FSDs operate independently, and each FSD searches maximum  $M$  candidate solutions. Hence, the PFSD algorithm reduces overall LLR value computation time by a factor of  $M_T$  compared to the LFSD and the  $K$ -best algorithms.

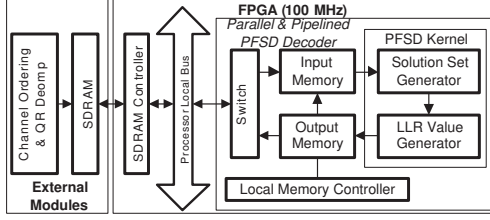
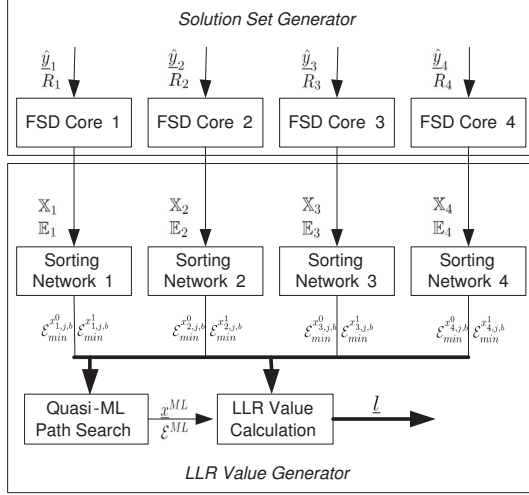
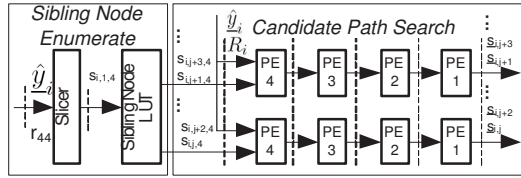


Fig. 3: The proposed architecture of PFSD.



(a)



(b)

Fig. 4: Block Diagram of: (a) PFSD kernel (b) FSD core.

## 5. FPGA IMPLEMENTATION OF PFSD

In this section, we discuss the hardware implementation details of the PFSD decoder for a  $4 \times 4$  MIMO system. This decoder is designed to support 4-QAM, 16-QAM and 64-QAM. The FPGA-based architecture for the PFSD algorithm with LLR calculation is shown in Figure 3. It consists of a SDRAM controller, a processor local bus (PLB), and a parallel and pipelined PFSD decoder. The decoder consists of the PFSD kernel, the input/output memory, and a local memory controller. The SDRAM controller fetches the channel preprocessing output data from the SDRAM and sends it to the input memory of the PFSD decoder. The PFSD decoder reads this data, processes it, and stores the results in the output memory. The SDRAM controller then reads these results and stores them back to the SDRAM. A 64-bit Xilinx Multi-Port Memory Controller (MPMC) is used to implement the SDRAM controller, and a 128-bit Process Local Bus (PLB)

is used to transfer data in and out of the PFSD decoder. The function of the PFSD decoder components is described below.

*Local Memory Controller:* This block controls the data access pattern of the PFSD kernel. The PFSD kernel requires 4 pairs of  $R_i$  and  $\hat{y}_i$  to generate one LLR vector  $\underline{l}$  (see Alg. 1 line 13). Each input is a 32-bit wide complex number with 16 bits for the real/imaginary parts. Each output LLR is 10-bit wide with 6 bits for fractional part. This bus can support 75Mbps throughput for  $4 \times 4$  64-QAM decoder running at 100MHz. For higher throughput decoder configurations, wider PLB bus must be used to meet data I/O requirement.

*Input/Output Memory:* There are 4 input memory and 1 output memory. All memories are dual-port, and each memory block is divided into three 128-bit wide 16-entry memory banks. Multiple banks in input memory blocks ensure parallel data access for the PFSD kernel.

*PFSD Kernel:* The block diagram of the PFSD kernel is shown in 4(a). The PFSD kernel consists of a solution set generator that performs parallel FSD tree search and a LLR value generator that runs the low complexity LLR algorithm. There are 4 parallel FSD cores, and each core has two function units—sibling node enumeration and candidate path search, as shown in Figure 4(b). The sibling node enumeration finds  $\hat{M}$  symbols expanded at the top level. It uses a slicer to determine the 1st symbol. Closest  $\hat{M}$  are found by using a LUT table. The candidate path search unit consists of two identical 4-stage pipeline connected processing elements (PE). It generates two outputs per clock cycle. PE1 to PE3 calculates  $\mathcal{V}_{i+1}$ , determines 1st child symbol  $s_i$  and calculating PED  $\mathcal{T}_i$ . PE4 only performs  $\mathcal{T}_i$  calculation. The LLR value generator consists of 4 sorting networks, a quasi-ML solution search module, and a LLR value calculation module. Each sorting network has 12 parallel 2 stage sorters. It finds  $(\mathcal{E}_{min}^{x_{i,j,b}^0}, \mathcal{E}_{min}^{x_{i,j,b}^1})$  for the  $i$ th FSD core, and save results in the LLR value calculation module. Quasi-ML path search module searches the overall minimum paths  $\underline{s}^{ML}$ , and updates the final LLR values.

The RTL code is synthesized for the Xilinx Virtex-4 (X4VLX160) device. Table 2 shows the total area and individual component utilization of the PFSD decoder. The percentage numbers for the PFSD total area entries are calculated with respect to the overall available FPGA resources. The FSD shared modules within the solution set generator include shared control signals and delay registers within each FSD core. They are less than 3% of the PFSD decoder. The local memory controller generates input and output memory address for FSD cores and the LLR value generator. It takes up less than 1% of the PFSD decoder. The soft-output PFSD implementation is a little more than 4 times larger than the hard output FSD implementation in [13], where 10745 4-input LUTs are used. Each PFSD is clocked at 100MHz; the critical path is in the sorting network, where current path metrics are compared with existing path metrics. For  $4 \times 4$  64-QAM decoder running at 100MHz, 24 LLRs are produced by the LLR value generator every 32 cycles. This translates

**Table 2:** FPGA Device Utilization Summary for PFSD and LLR Calculation (at 100MHz, 4-FSD cores)

Xilinx XC4VLX160		Device Utilization				
		Slice Flip Flops	4 Input LUTs	Slices	RAMB16	DSP48s
PFSD Total Area		32464 (24%)	46325 (34%)	25787 (38%)	48 (16%)	64 (66%)
Solution Set Generator	FSD Cores	26280	38060	20256	12	64
	FSD Shared	1304	1016	680		
LLR Value Generator	Sorting Network	3904	5148	3672		
Local Memory Controller	Quasi-ML Search	352	375	271	4	
	LLR Value Calc	504	1512	768		
Input Memory					16	
Output Memory					16	

to a throughput of 75Mbps.

The ASIC implementation of K-best decoder [3] achieves 107Mbps throughput for 4x4 16-QAM system at a clock rate of 200MHz. For the same MIMO system configuration, PFSD decoder achieves 200Mbps throughput at a clock rate of 100MHz. For 4x4 64-QAM system, the ASIC implementation of FSD in [4] achieves 215Mbps throughput at a clock rate of 574.7MHz. In theory, the PFSD decoder can achieve same throughput if clocked at 287MHz. However in the current FPGA platform, the PFSD decoder only runs at 100MHz. To achieve 300Mbps, additional 4 PEs can be added to each FSD core, which results in 2x increase in decoder size. In general, for higher throughput, additional PEs can be added to the candidate path search unit in a FSD core, resulting in higher number of candidate vector outputs per clock cycle.

## 6. CONCLUSION

In this paper, we developed a high throughput parallel fixed complexity sphere decoding (PFSD) algorithm. We also designed a low complexity parallel QR decomposition that reduces the PFSD channel preprocessing overhead. The PFSD provides high bit diversity for each received signal component and simplifies the child node enumeration step that is required in the existing soft-output sphere decoders. Through simulation, we demonstrate that the PFSD algorithm with 32 candidates has lower computation complexity and comparable performance to LFSD and  $K$ -best in a  $4 \times 4$  16-QAM system. Furthermore, the PFSD algorithm is about  $M_T$  times faster than LFSD and  $K$ -best since it can be mapped onto  $M_T$  parallel FSDs. The PFSD algorithm has been implemented on Xilinx VC4VLX160 FPGA. For  $4 \times 4$  64-QAM configuration, the PFSD decoder can achieve 75Mbps running at 100MHz.

## 7. REFERENCES

- [1] G.J. Foschini, "Layered space-time architecture for wireless communication in a fading environment when using multi-element antenna," *Bell Labs Technical Journal*, vol. 1, no. 2, pp. 41–59, 1996.
- [2] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," in *SIGSAM Bull.*, Feb. 1981, vol. 15, pp. 37–44.
- [3] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection," *IEEE Transactions on Selected Areas in Communications*, vol. 44, no. 3, pp. 491–503, March 2006.
- [4] R. Bhagawat, P. Dash and G. Choi, "Systolic like soft-detection architecture for 4x4 64-QAM MIMO system," in *IEEE DATE 2009*, pp. 870–873.
- [5] L.G. Barbero and J.S. Thompson, "Extending a fixed-complexity sphere decoder to obtain likelihood information for Turbo-MIMO systems," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 5, pp. 2804–2814, Sept. 2008.
- [6] C.P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," in *Fundamentals of Computation Theory*, 1991, pp. 68–85.
- [7] K.W. Wong and et al., "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *IEEE ISCAS*, 2002, vol. 3, pp. 273–276.
- [8] L.G. Barbero and J.S. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," in *IEEE SPAWC 2006*, pp. 1–5.
- [9] C. Hess and et al., "Reduced-complexity MIMO detector with close-to-ML error rate performance," in *ACM Great Lakes Symposium on VLSI*, March 2007, pp. 200–203.
- [10] M. Li and et al., "An implementation friendly low complexity multiplierless LLR generator for soft MIMO sphere decoders," in *IEEE SIPS 2008*, Oct., pp. 118–123.
- [11] M. Sitti and M.P. Fitz, "A novel soft-output layered orthogonal lattice detector for multiple antenna communications," in *IEEE ICC 2006*, pp. 1686–1691.
- [12] B.M. Hochwald and S. Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Transactions on Communications*, vol. 51, no. 3, pp. 389–399, March 2003.
- [13] R. Bhagawat, P. Dash and G. Choi, "Architecture for reconfigurable MIMO detector and its FPGA implementation," in *IEEE ICECS 2008*, pp. 61–64.