J Sign Process Syst (2012) 68:217–231
DOI 10.1007/s11265-011-0602-1

Parallel High Throughput Soft-Output Sphere Decoding Algorithm

Qi Qi · Chaitali Chakrabarti

Received: 6 October 2010 / Revised: 9 June 2011 / Accepted: 9 June 2011 / Published online: 7 July 2011

Abstract Multiple-Input-Multiple-Output communication systems demand fast sphere decoding with high performance. To speed up the computation, we propose a scheme with multiple fixed complexity sphere decoders to construct a parallel soft-output fixed complexity sphere decoder (PFSD). The proposed decoder is highly parallel and has performance comparable to soft-output 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 that the PFSD algorithm can increase the throughput and reduce bit error rate of a soft-output solution in a 4×4 16-QAM system, and has superior performance compared to other soft decoders with comparable throughput and computation complexity. The PFSD algorithm has been mapped onto Xilinx XC4VLX160 FPGA. The resulting PFSD decoder can achieve up to 75 Mbps throughput for 4×4 64-QAM configuration at 100MHz with low control overhead.

Keywords Soft-output sphere decoding · Parallel algorithm · Fixed complexity

Q. Qi (✉) · C. Chakrabarti
School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287-5706, USA
e-mail: qi@asu.edu

C. Chakrabarti
e-mail: chaitali@asu.edu

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. The signal detector is one of the most important modules in a MIMO system. Maximum-likelihood (ML) 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) [2], semidefinite relaxation (SR) [3, 4] and sphere decoding (SD) [5]. 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 [6].

The soft-output SD algorithm is favored over the hard output SD algorithm due to its significant performance gain in low and medium SNR conditions [7]. 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 [7, 8], 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 [9]. However, the LLR generator involves sorting large number of candidate solutions, and thereby 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 find that for a 4×4 16-QAM system, the PFSD provides better BER performance than the list fixed complexity decoder (LFSD) [10].
2. We use a *parallel QR decomposition* algorithm for PFSD that shares intermediate results from multiple QR decompositions. As a result, the throughput of this step can be increased by 100% compared to serial QR decomposition, with minimal addition of computation units.
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. It can deliver up to 400 Mbps, 200 Mbps and 75 Mbps throughput for 4×4 systems with 4-, 16- and 64-QAM configuration when clocked at 100 MHz.

The rest of the paper is organized as follows. We briefly describe a MIMO system in Section 2, followed by a review of the sphere decoding algorithm in Section 3. Section 4 presents the PFSD, the parallel QR decomposition and the low complexity LLR algorithms. Section 5 provides algorithm simulation results. Section 6 discusses the hardware architecture for PFSD. The conclusion is given in Section 7.

2 Preliminaries

A basic MIMO Bit-interleaved coded modulation (BICM) communication system [11] consists of channel encoder, interleaver ($\prod$), modulation and mapping unit, demodulation and MIMO detector, de-interleaver ($\prod^{-1}$) and channel decoder, as shown in Fig. 1. Assume that there are M_T transmit antennas and M_R receive antennas. Let $\underline{\tilde{u}}$ denote a vector of uncoded source data bits that is input to a channel encoder of rate $R\leq1$ to produce a coded bit vector $\underline{\tilde{c}}$. The channel encoder output vector $\underline{\tilde{c}}$ is interleaved to obtain bit vector $\underline{\tilde{x}}$. Let $\underline{\tilde{s}}$ be an $M_R\times1$ vector of transmitted symbols. Each transmitted symbol is obtained by mapping every $M_c=\log_2(M)$ consecutive bits from $\underline{\tilde{x}}$ onto an M-symbol

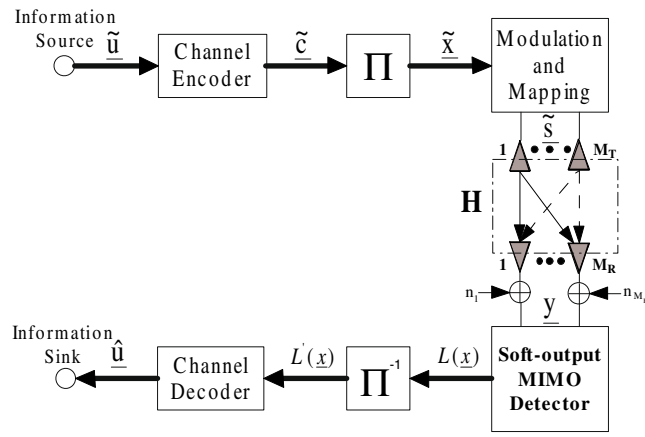


Figure 1 Block diagram of a MIMO communication system.

constellation. Let $\underline{y}$ be the $M_R\times1$ vector of received symbols, given by

$$\underline{y}=H\underline{\tilde{s}}+\underline{n} \tag{1}$$

where H is an $M_R\times M_T$ complex channel matrix, and $\underline{n}$ is an $M_R\times1$ noise vector. Each element h_{ij} in H represents the complex transfer function from the jth transmit antenna to the ith receive antenna. All elements in H are independent and identically distributed (i.i.d.) complex Gaussian variables with zero mean and variance 1. Similarly, all elements in n are i.i.d. complex Gaussian variables with zero mean and σ^2 variance, where σ^2 is calculated according to the received SNR.

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

$$\underline{s}_{SD}=\arg\min_{\underline{s}\in\mathbb{O}\backslash\mathbb{S}}\parallel\underline{y}-H\underline{s}\parallel^2,\ \underline{s}_{SD}\in\mathbb{S} \tag{2}$$

where $\mathbb{O}=M^{M_T}$ denotes all possible transmitted symbol vectors which reside in an M_T-dimensional square lattice spanned by an M complex constellation in each dimension. A set of possible transmitted coded bit vectors $\{\underline{x}\}$ is obtained by demodulating symbol set $\{\underline{s}_{SD}\}$. The channel decoder then uses the aposteriori log-likelihood ratio value (LLR) of the bits from de-interleaved $\{\underline{x}\}$ to calculate the likely transmitted data sequence $\underline{\hat{u}}$.

For any received vector $\underline{y}$, the low complexity *Max-log approximation* LLR $L(x_k|\underline{y})$ for bit x_k is calculated as

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

where $\underline{s}$ is an $M_T\times1$ symbol vector and $\underline{x}$ is the corresponding $M_TM_c\times1$ bit vector. Set $\mathbb{X}_{k,+1}$ and $\mathbb{X}_{k,-1}$ denote subsets of $\{\underline{x}\}$ with the kth bit $x_k=+1$ and $x_k=-1$, respectively. Hence, the kth bit LLR of $\underline{x}$ requires two

bit vectors with opposite binary values in the kth bit position. One of the bit vectors is always the ML solution. The other bit vector is the complementary solution, whose kth bit is defined as the *counter-hypothesis* bit [12]. The distance between the hypothesis bit solution and counter-hypothesis bit solution is a measure of the reliability of the bit value.

Throughout this paper, we assume that the system is full rank, where $M_R \geq M_T$. The channel matrix H and the noise variance σ^2 are known to the receiver through training sequence estimation. To generate the maximum likelihood solution, one has to search over M^{M_T} symbol vectors. Even for a moderate 64-QAM 4×4 system, the complete symbol set $\{\underline{s}\}$ contains 16, 777, 216 candidates. To avoid exhaustive search, approximate LLRs are calculated by using soft sphere decoding (SD) algorithm, which finds the K smallest norm solutions to Eq. 2 in $\{\underline{s}_{SD}\}$, where $K \ll M^{M_T}$.

3 Sphere Decoding Algorithm

SD algorithms enumerate vector solutions $\underline{s}$ of Eq. 2, where $\underline{s} \in \mathbb{S} \subset \mathbb{O}$, and $\underline{s}$ is inside an ellipsoid centered at $\hat{y}$. To efficiently search the ML solution inside the ellipsoid, a SD algorithm converts the original least square problem to a tree search problem using the following steps. It first performs the QR decomposition of the channel matrix H, where $H = QR$. The resulting Q is an $M_R \times M_R$ orthogonal matrix, and R is an $M_R \times M_T$ nonsingular upper triangular matrix [6]. Equation 2 undergoes orthogonal transformation, and can be represented as

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

where $\underline{\hat{y}} = Q^H \underline{y} = R\underline{s}^{ZF}$. Q^H denotes the complex conjugate transpose of Q, $\underline{s}^{ZF} = R^{-1}Q^H\underline{y} = H^{\dagger}\underline{y}$ is the zero-forcing (ZF) solution, and $H^{\dagger}$ is the pseudo-inverse of H. We use $\mathcal{E}$ to represent the squared distance between estimated and transmitted signals.

Due to the upper triangular nature of matrix R, Eq. 4 can be further recursively decomposed to generate the following expressions

$$\begin{aligned} \| \underline{\hat{y}} - R\underline{s} \|^2 &= \sum_{i=M_T}^{1} \left| \left[\hat{y}_i - \sum_{j=M_T}^{i+1} r_{ij}s_j \right] - r_{ii}s_i \right|^2 \\ &= \sum_{i=M_T}^{1} \left| \mathcal{V}_{i+1}\left(\underline{s}_{(i+1)}\right) - r_{ii}s_i \right|^2 \\ &= \sum_{i=M_T}^{1} \left| \mathcal{D}_i\left(\underline{s}_{(i)}\right) \right|^2 \end{aligned} \tag{5}$$

where $\underline{s}_{(i)} = [s_i\, s_{i+1} \ldots s_{M_T}]^T$ denotes a *partial vector symbol* candidate, $\mathcal{V}_{i+1}$ is the corresponding *residual vector metric* and $\mathcal{D}_i(\underline{s}_{(i)})$ is the respective *branch metric*. The graphic illustration of Eq. 5 is an inverse $M_T + 1$ level M-ary tree, where each node has M child nodes except the leaf nodes on level 1. Each branch is associated with a branch metric which is always positive. The *partial Euclidean distance metric* (PED), $\mathcal{T}_i = \sum_{i=M_T}^{i} | \mathcal{D}_i(s_{(i)}) |^2$, is the accumulative branch metrics along a path from the root node to a node at level i, and is *monotonically non-decreasing*. Note that the sibling nodes have identical PED value $\mathcal{T}_{i+1}$ and residual vector metric $\mathcal{V}_{i+1}$.

Existing SD algorithms are based on either the *depth-first* or *breadth-first* search. The depth-first SD algorithms, such as the Fincke–Pohst (F–P) algorithm [13] and the Schnorr–Euchner (SE) algorithm [14], generate one candidate solution at a time, and reduce the search radius r based on the current best solution. The K-best algorithm [15] is a type of breadth-first SD algorithm, where K candidate paths are generated at a time. From hardware implementation stand-point, the K-best algorithm has several advantages over the depth-first algorithms, including fixed decoder state transition, parallel child node extension, and constant throughput. However, a K-best based algorithm needs to find the K smallest PED paths out of KM candidate paths at each decoding level, which requires large number of sorting operations for moderate K and M. This issue is circumvented by fixed complexity SD algorithms [16–19].

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 $\underline{t} = [t_1, t_2, \ldots, t_{M_T}]$. At level i, each parent node enumerates t_i child nodes in increasing order of their branch metrics. Fixed complexity SD algorithms eliminate the sorting procedure by keeping all expanded nodes. The resulting number of partial vector candidate $\underline{s}_{(i)}$ is $\prod_{j=i}^{M_T} t_j$, where $1 \leq t_j \leq M$. The cardinality vector $\underline{t}$ and symbol detection order can greatly impact the performance of a fixed complexity SD algorithm. The algorithm in [16] offers an effective solution, and it is denoted as the fixed-complexity SD (FSD) algorithm in the following sections.

The FSD algorithm consists of three steps, namely, *channel matrix ordering*, *solution set generation* and *hard decision selection*. Channel matrix ordering determines the order in which the symbols in vector $\underline{s}$ are detected. Solution set generation finds solution candidate vectors according to a predefined n. Hard decision selection finds the candidate vector with the smallest

PED value. FSD 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 branch metric is kept. The top p levels of solution set generation are of type FS, and the remaining $M_T - p$ levels are of type SS. The symbols in consecutive FS levels are detected in ascending order of their post-detection noise amplification, and the symbols in consecutive SS levels are detected in descending order of their post-detection noise amplification.

We developed a soft-output parallel fixed complexity SD (PFSD) algorithm based on the hard-output FSD algorithm, which not only delivers high throughput but also offers good scalability. The PFSD algorithm details are described in the following section. A summary of the key notations that are used in this paper is included in Table 1 for easy reference.

Table 1 Definition of key notations.

Notation	Definition
$\mathcal{D}_i$	Branch metric at level i
$\mathcal{E}$	Distance between estimated and transmitted signals
$\mathbb{E}_i$	Path metric set for $\mathbb{S}_i$
$\mathcal{E}_{\min}^{x_{i,b}^0}$	Minimum path metric for the bth bit in $\mathbb{X}_i$ with value 0
$H^\dagger$	Pseudo-inverse of H
H_i	Column-wise permuted H according to $\underline{k}_i$
$\underline{k}_i$	ith permutation vector
$k_{i,l}$	lth element of $\underline{k}_i$
$\underline{l}$	LLR value vector for $\underline{s}$
M	QAM constellation size
$\hat{M}$	Cardinality value for t_{M_T}
M_c	Bits per M-QAM symbol
$\underline{t}$	Child node cardinality vector
n_j	Child nodes per parent node at level j
Q	Orthogonal matrix
$q_{j,l}$	lth column of Q_j
R	Upper triangular matrix
r	Sphere decoder search radius
$\underline{s}$	Candidate symbol vector
$\mathbb{S}$	A set of candidate symbol vectors
$\mathbb{S}_i$	A subset of $\mathbb{S}$ generated from H_i and $\underline{\hat{y}}_i$
s_i	A candidate symbol at level i
$\underline{s}_{(i)}$	Partial vector symbols contains elements s_i to s_{M_T}
$\underline{s}^{ML}$	Maximum likelihood symbol vector
$\underline{s}^{ZF}$	Zero forcing solution
$\underline{s}_{i,j}$	jth symbol vector in $\mathbb{S}_i$
s_{i,j,M_T}	M_Tth symbol of jth symbol vector in $\mathbb{S}_i$
$\mathcal{T}_i$	Partial distance metric from root node to a node at level i
$\mathcal{V}_i$	Residual vector metric for level i
$\underline{x}$	Candidate bit vector
$\mathbb{X}_i$	Bit vector equivalence of $\mathbb{S}_i$
$x_{i,b,j}^0$	bth bit of the jth bit vector in $\mathbb{X}_i$ with bit value of 0
$\underline{y}$	Received symbol vector
$\underline{\hat{y}}_i$	Orthogonal transformation of $\underline{y}$

4 Parallel Fixed Complexity Sphere Decoding

The fixed complexity SD algorithms are best suited for VLSI implementation due to simplification in path pruning in the solution generation steps [18]. However, existing soft decision fixed complexity sphere decoding algorithms [10, 20] provide high diversity in the bit values by either selective child node enumeration, or expansion of larger number of child nodes in successive levels. Both procedures introduce data dependency between sibling nodes, and require additional computation. We propose a high throughput parallel fixed complexity sphere decoding (PFSD) algorithm which eliminates this dependency by producing soft decision outputs from multiple hard decision FSDs. The PFSD algorithm provides good BER performance when compared to a LFSD algorithm with similar computation complexity. In addition, the PFSD algorithm make use of layer ordering technique in [21] to reduce preprocessing overhead. Last, the PFSD algorithm produces more reliable LLR values with slight increase in computation overhead than a competing parallel fixed complexity sphere decoding algorithm, Layered Orthogonal Decoding (LORD) in [22].

The proposed PFSD algorithm is presented in Algorithm 1. It starts with the same channel matrix ordering step as the FSD algorithm in [16] and 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 permutation of the original H according to $\underline{k}_i$. In the solution set generation step, PFSD produces candidate set $\mathbb{S}_i$ for each pair of H_i and y by performing FSD search with cardinality vector $\underline{t}_i = [1, 1, \ldots, \hat{M}]$ (lines 9–15). 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 5. 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 and 20). The LORD algorithm also uses multiple FSDs to generate candidate solution set. However, it differs in the way multiple QR decomposition are done (Section 4.1) as well as the way LLR values are calculated (Section 4.2).

Algorithm 1 Soft decision PFSD algorithm

Require: M, M_T, $\underline{y}$, H
{Channel Matrix Ordering:}
Produce permutation vector $\underline{k}_1 = [k_{1,1}, k_{1,2}, \ldots, k_{1,M_T}]$
for $i = 2$ to M_T **do**
  Construct $\underline{k}_i$, where $\underline{k}_{i,M_T} \neq \underline{k}_{j,M_T}, \forall j < i$
end for
{Solution Set Generation:}
Set $\mathbb{S} = \emptyset$
for $i = 1$ to M_T **do**
  Generate H_i by permuting H column-wise according to $\underline{k}_i$
  $[Q_i, R_i] = qr(H_i)$, $\hat{\underline{y}}_i = Q_i^H \underline{y}$
  $\underline{t}_i = [1, \ldots, 1, \hat{M}]$, $\hat{M} \leq M$
  {FSD tree search:}
  Input: R_i, $\hat{\underline{y}}_i$, $\underline{t}_i$; Output: Solution subset $\mathbb{S}_i$
  Inverse permute candidate vectors in $\mathbb{S}_i$ using $\underline{k}_i$
  $\mathbb{S} = \mathbb{S}_i \cup \mathbb{S}$
end for
{Calculate Outputs:}
Assign minimum weight vector in $\mathbb{S}$ to $\underline{s}^{ML}$
Calculate LLR values $\underline{l}$ from $\mathbb{S}$
return hard and soft decision outputs

Essentially, PFSD produces soft decision outputs by performing multiple hard decision FSDs. The theoretical analysis in [23] proves that FSD can provide the same diversity order as the ML decoder, and yield asymptotically ML performance in the high SNR region under the following condition

$$(M_R - M_T)(q+1) + (q+1)^2 > M_R \tag{6}$$

where q is the number of FS levels. For a 4×4 MIMO system, 1 FS level is sufficient ($\underline{t}_i = [1, \ldots, 1, M]$). The symbol detection order of each FSD in PFSD is uniquely determined by its permutation vector. Element k_{i,M_T} specifies the 1st detected symbol of the ith FSD. Since k_{i,M_T} differs from k_{j,M_T} provided $i \neq j$, each symbol element of vector $\underline{s}$ becomes the 1st detected symbol exactly once. In case partial expansion is used, where $t_{M_T} = \hat{M} < M$, $\hat{M}$ must be sufficiently large to guarantee the existence of counter-hypothesis bit at each bit position. This condition improves the reliability of LLRs. Note that the number of candidate nodes at each level is uniform, and the number of calculations involved in finding all candidate vectors is identical.

The complexity analysis of PFSD includes that of the channel matrix ordering step and the tree search step. The channel matrix ordering step is identical to the FSD algorithm. Since the tree search step is equivalent to multiple FSD decoding, the computation complexity of the tree search step is simply M_T times the complexity of one stage full expansion FSD implementation. However, the PFSD requires multiple QR decompositions, which adds computation complexity to the overall algorithm. In the next subsection, we present a parallel QR decomposition algorithm that can generate outputs of two H matrices every cycle, thereby reducing the overhead. The multiplication complexity of the tree search step, without including the QR decomposition overhead, is given by

$$N_{\text{mult}} = M_T \sum_{i=1}^{M_T} \left[aM + (M_T - i) b\, M \right] \tag{7}$$

where a denotes the number of real multiplications for l^2-norm calculations of branch metric (a is typically 2), and b denotes the number of real multiplications for a complex product (b is typically 3 or 4 depending on the implementation). It is easy to see that N_{mult} only depends on M_T and M. For a given modulation scheme, N_{mult} increases cubically with M_T; for a fixed antenna system, N_{mult} increases linearly with respect to the modulation size.

4.1 Parallel QR Decomposition for PFSD

To support multiple tree searches in PFSD, a QR decomposition is needed for each FSD. We present a parallel QR decomposition algorithm based on [21] that lowers channel processing overhead by combining the channel matrix ordering with QR decomposition. For instance, for a 4×4 system, if matrices H_i and H_j differ only in the two right most columns, the common intermediate results generated during their QR decompositions can be shared, resulting in significant computation saving. The permutation vectors for 4×4 serial and parallel QR decompositions are illustrated in Fig. 2a. Alphabets *a–d* are used to denote the values of permutation vector $\underline{k}_j$, $1<j<4$. In parallel QR, for vector $\underline{k}_1$ and $\underline{k}_2$, the first two columns of H_1 and H_2 are the same. Similarly for vectors k_3 and k_4, the first two columns of H_3 and H_4 are the same. Hence, they share identical decomposition results, and redundant computation steps can be avoided. Figure 2b shows the permutation vectors for 8×8 system. For example, first four columns of H_1–H_4 (H_5 through H_8) share identical decomposition results, and their permutation vector elements *a–d* (*h* through *g*) have same color filling. Finally, existing QR decomposition systems are based on Given Rotations [24] and Householder transformation [25]. Unfortunately, those methods cannot

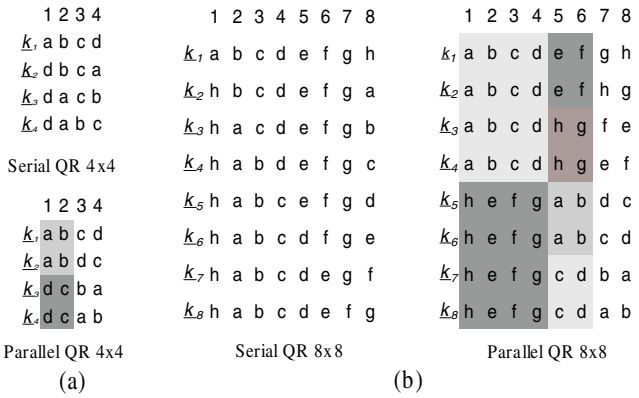


Figure 2 Permutation order comparison of serial QR and parallel QR decompositions for **a** 4×4 and **b** 8×8 channel matrices.

Algorithm 2 Parallel QR Decomposition algorithm

1: {Channel Matrix Order (optional)}
2: Perform improved sorted Gram-Schmidt QR decomposition [26] for H_1
3: Generate permutation vector $\underline{k}_1$
4: **for** $j = 2$ to M_T **do**
5: Generate permutation vector $\underline{k}_j$ from $\underline{k}_{j-1}$ {See parallel QR channel matrix examples in Fig. 3}
6: Obtain H_j by column-wise permuting H according to $\underline{k}_j$
7: Perform QR decomposition on H_j {Copying shared intermediate results from previous QR}
8: **end for**

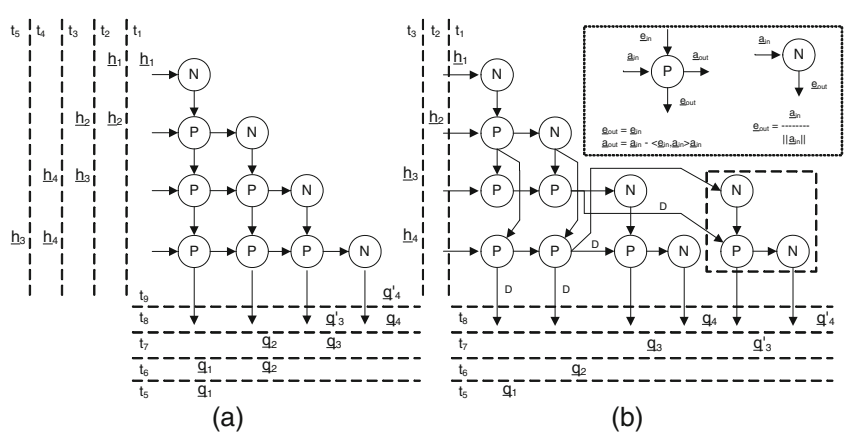
be easily extended to exploit the common computations and result in large overhead.

The parallel QR decomposition is shown in Algorithm 2. We use the improved sorted Gram-Schmidt QR decomposition algorithm in [26], where the 2nd minimum norm square $|q_j|^2$ is chosen for k_i for $i \geq 2$. As a result, H_1 satisfies the channel matrix ordering for FSD but incurs additional compare and vector l^2 norm operations. This is useful for hard decision decoding under very high SNR conditions, where the minimum weight solution generated from H_1 is almost always the transmitted vector. However, simulation results in Section 5.1 show that channel matrix ordering step has negligible performance impact on PFSD, and can be excluded to reduce computation complexity.

Figure 3 shows the data flow graphs (DFG) of Gram-Schmidt method for both serial and parallel QR decomposition. In both cases, two 4×4 channel matrices $H_1 = [\underline{h}_1, \underline{h}_2, \underline{h}_3, \underline{h}_4]$ and $H_2 = [\underline{h}_1, \underline{h}_2, \underline{h}_4, \underline{h}_3]$ are fed as inputs, and two orthonormal matrices $Q_1 = [\underline{q}_1, \underline{q}_2, \underline{q}_3, \underline{q}_4]$ and $Q_2 = [\underline{q}_1, \underline{q}_2, \underline{q}'_3, \underline{q}'_4]$ are produced as outputs. Both DFGs consist of two types of nodes, N and P. Node N takes an input vector $\underline{a}_{\text{in}}$ and produces a normalized basis vector $\underline{e}_{\text{out}}$ which is passed to the P node below. Node P has two inputs, $\underline{a}_{\text{in}}$ and $\underline{e}_{\text{in}}$, and two outputs, $\underline{a}_{\text{out}}$ and $\underline{e}_{\text{out}}$. $\underline{e}_{\text{out}}$ is set to $\underline{e}_{\text{in}}$ and passed to the P below; $\underline{a}_{\text{out}}$ is computed by subtracting the subspace component of $\underline{a}_{\text{in}}$ in the direction of basis $\underline{e}_{\text{in}}$, and passed to the node to its right. All input and output vectors are complex. The elements in the R matrix (of the QR decomposition) are a by-product of the computation in the P and N nodes. Essentially, r_{ij} in R is the Hermitian inner product between $\underline{e}_{\text{in},i}$ and $\underline{a}_{\text{in},j}$.

For the 4×4 system, node P requires 24 real multiplications, 47 real additions, and node N requires 16 real multiplications, 7 real additions, 1 real division and 1 real square-root. If we assume that each node takes 1 cycle, then the parallel QR produces the outputs of Q_1 and Q_2 in cycles 5–8. Since it is a pipelined implementation, the next set of outputs (corresponding to Q_3 and Q_4) can be produced in cycles 6–9. In contrast, the pipelined implementation of the serial QR produces outputs of Q_1 and Q_2 in cycles 5–9 and outputs of Q_3

Figure 3 Data flow graphs for: **a** serial QR and **b** parallel QR decompositions for 4×4 channel matrix.



and Q_4 in cycles 7–11. Note there is a 3 cycle latency overhead for the 1st QR decomposition and after that the parallel QR produces outputs at twice the rate as the serial QR. For $M_R \times M_R$ matrix, the parallel QR only requires one additional P node and two additional N nodes more than the serial QR to achieve the 100% throughput gain.

The parallel QR algorithm not only has reduced the computation complexity, but also has lower memory storage requirement compared to performing each QR decomposition independently in parallel. The two channel matrices H_i and H_j derived from permuting channel matrix H, differ in last two columns. The QR decomposition results of H_i and H_j show that Q_i and Q_j differ in the last two columns and R_i and R_j differ in last two rows. Hence, memory requirement of the parallel QR decomposition results does not increase linearly with respect to antenna size. For the 4×4 system, the memory requirement of the parallel QR algorithm is only 2.8X more than a single QR decomposition, and is 30% less than performing each QR decomposition independently in parallel. For the 8×8 system, the memory requirement of the parallel QR algorithm is 3.6X more than a single QR decomposition, and is 55% less than performing each QR decomposition independently in parallel.

Algorithm 3 Low complexity LLR algorithm

Require: $\{\underline{k}_i\}, \mathbb{E}_i, \mathbb{S}_i, \forall i=1,\ldots,M_T$

1: $\mathbb{X}_i \xleftarrow{\text{demod}(s_{i,j,M_T})} \mathbb{S}_i, \forall i=1,\ldots,M_T$
2: $\underline{s}^{ML}=[\,], \underline{l}^0=[\,], \underline{l}^1=[\,], \mathcal{E}^{ML}=\infty$
3: **for** $i=1$ to M_T **do** {for $\mathbb{X}_i$}
4: **for** $b=1$ to M_c **do** {for $x_{i,b,j}$}
5: **if** $x_{i,b,j}==0$ **then**
6: $\mathcal{E}^{x^0_{i,b}}_{\min}=\min_{j=1,\ldots,\hat{M}} \mathcal{E}^{x^0_{i,b,j}}, \mathcal{E}^{x^0_{i,b,j}}\in\mathbb{E}_i, x^0_{i,b,j}\in\mathbb{X}_i$
7: add $\mathcal{E}^{x^0_{i,b}}_{\min}$ to $\underline{l}^0$
8: **else**
9: $\mathcal{E}^{x^1_{i,b}}_{\min}=\min_{j=1,\ldots,\hat{M}} \mathcal{E}^{x^1_{i,b,j}}, \mathcal{E}^{x^1_{i,b,j}}\in\mathbb{E}_i, x^1_{i,b,j}\in\mathbb{X}_i$
10: add $\mathcal{E}^{x^1_{i,b}}_{\min}$ to $\underline{l}^1$
11: **end if**
12: $\mathcal{E}^{ML}=\min(\mathcal{E}^{x^0_{i,b}}_{\min}, \mathcal{E}^{x^1_{i,b}}_{\min}, \mathcal{E}^{ML})$
13: $\underline{k}^{ML}=\underline{k}_i, \underline{s}^{ML}=\underline{s}^{\mathcal{E}^{ML}}, \underline{s}^{\mathcal{E}^{ML}}\in\mathbb{S}^i$, if $\mathcal{E}^{ML}$ changes
14: **end for**
15: **end for**
16: inverse permute $\underline{s}^{ML}$ using $\underline{k}^{ML}$
17: $\underline{x}^{ML}=\text{demod}(\underline{s}^{ML})$
18: update $\underline{l}^0$ and $\underline{l}^1$ using $\underline{x}^{ML}$ and $\mathcal{E}^{ML}$
19: calculate LLR $\underline{l}=\underline{l}^0-\underline{l}^1$
20: **return** $\underline{s}^{ML}, \mathcal{E}^{ML}$ and LLR values

4.2 Low Complexity LLR Algorithm for PFSD

Generating LLR values from a large list of candidate symbol vectors contribute significantly to the computation complexity to the overall soft-output decoding. The proposed low complexity LLR algorithm for PFSD produces suboptimal soft-output values for a given set of candidate vectors in order to reduce the number of sorting operations. Equation 3 shows that to calculate the LLR for one bit position, we need the the quasi-ML solution, and the counter-hypothesis bit solution. The low complexity LLR algorithm searches all candidate vectors for the quasi-ML solution. However, it only searches a subset of candidate vectors for each counter-hypothesis bit solution. Furthermore, since candidate vector subsets are independent from each other, LLR values can be computed in parallel without compromising the quasi-ML solution. The proposed algorithm is summarized in Algorithm 3.

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 $\mathcal{E}^{x^0_{i,b}}_{\min}$ and $\mathcal{E}^{x^1_{i,b}}_{\min}$ within $\mathbb{E}_i$. The $\hat{M}(M_T-1)$ symbol candidate vectors from other subsets are ignored. However, the quasi-ML solution is selected from all $\hat{M}M_T$ vectors. The ith bit vector subset $\mathbb{X}_i$ is obtained by only demodulating s_{i,j,M_T} in $\mathbb{S}_i$ (see line 1). Variable s_{i,j,M_T} denotes the M_Tth symbol element of the jth symbol vector, where $1\le j\le\hat{M}$. The algorithm then searches and saves minimum path metrics $\mathcal{E}^{x^0_{i,b}}_{\min}$ and $\mathcal{E}^{x^1_{i,b}}_{\min}$ associated with subset $\mathbb{X}_i$ (see lines 5–11). The bth bit of the jth bit vector is denoted by $x_{i,b,j}$, where $1\le b\le M_c$; $x^0_{i,b,j}$ and $x^1_{i,b,j}$ represents $x_{i,b,j}$ with bit value 0 and 1 respectively, and their PED values are denoted by $\mathcal{E}^{x^0_{i,b,j}}$ and $\mathcal{E}^{x^1_{i,b,j}}$. Vector $\underline{l}^0$ and $\underline{l}^1$ records newly found $\mathcal{E}^{x^0_{i,b}}_{\min}$ and $\mathcal{E}^{x^1_{i,b}}_{\min}$. The quasi-ML solution $\underline{s}^{ML}$ is updated along with its permutation vector $\underline{k}^{ML}$ when either $\mathcal{E}^{x^0_{i,b}}_{\min}$ or $\mathcal{E}^{x^1_{i,b}}_{\min}$ is smaller than its current path metric $\mathcal{E}^{ML}$ (see lines 12 and 13). LLR values $\underline{l}$ are finally calculated using properly permuted $\underline{s}^{ML}$, l^0 and l^1 (see lines 16–19).

The low complexity LLR algorithm performs suboptimal search of the counter-hypothesis bits, and full search of the quasi-ML bits. Since $\mathcal{E}^{x^0_{i,b}}_{\min}$ and $\mathcal{E}^{x^1_{i,b}}_{\min}$ are generated locally for the M_Tth symbol within the ith FSD, the number of compare operations can be reduced by exploiting the property of QAM symbols. Recall that symbols in the same column share identical in-phase

binary code and symbols in the same row share identical quadrature binary code. Hence, path metrics for symbols with identical in-phase and quadrature components are minimized first. Then, minimum path metrics $\mathcal{E}_{\min}^{x_{i,b}^0}$ and $\mathcal{E}_{\min}^{x_{i,b}^1}$ are generated for the bth bit position from surviving path metrics, which also require $2\lceil\sqrt{\hat{M}}\rceil$ symbol demodulations. The total number of compare operations for the low complexity LLR algorithm is given by

$$N_{\text{comp}} = \left[2M_T\left(\tilde{M}-1\right)\tilde{M}\right] + \left[M_T M_c\left(\tilde{M}-2\right)\right] + [2M_T - 1] \quad (8)$$

where $\tilde{M} = \lceil\sqrt{\hat{M}}\rceil$, $\hat{M} \leq M$. The first term in Eq. 8 calculates the number of quadrature and in-phase symbol path metric comparisons. The second term calculates the number of bit position path metric comparisons. The last term counts the additional compare operations for the quasi-ML solution. Parallel implementation of the compare operations require $2M_T\tilde{M}$ sorting networks of size M_T, $M_T M_c$ sorting networks of size $(\tilde{M}-1)$ and 1 sorting network of size $2M_T$. For a 4×4 system, the sorting networks are not very large, and yet they contribute to 14% of the decoder area in our FPGA implementation!

Existing methods such as K-best and LFSD are based on full search of counter-hypothesis bits. When all $\hat{M}M_T$ symbol candidate vectors are considered in a full counter-hypothesis bit search, additional $M_T M_c[(M_T-1)\hat{M}]$ compare operations are required to find the bit values. For a 4×4 system, this translates to 85% and 88% less compare operations compared to the full search algorithm for 16-QAM and 64-QAM, respectively.

5 Simulation Results

In this section, we compare the Monte-Carlo simulation results of PFSD, LFSD and K-best sphere decoding algorithms. The system under investigation consists of $M_T=4$ transmit and $M_R=4$ receive antennas. The uncoded source data vector $\underline{\tilde{u}}$ is 4,096 bits long. It is encoded by a parallel concatenated Turbo encoder, which consists of two component $R_c=1/2$ Recursive Systematic Convolutional (RSC) component encoders. The generator matrices for the encoders are $g1(D) = 1 + D + D^2$ and $g2(D) = 1 + D^2$ [27]. The output of the encoder $\underline{\tilde{c}}$ is a 8,192 bits long packet. Each packet is passed through a pseudo-random interleaver, and reshaped to a $4\times2{,}048$ bit matrix. Gray code mapping is used to map every $M_c = 4$ consecutive bits in each row to a 16-QAM symbol. A vector of 4 symbols is transmitted at a time, one from each antenna. Block Rayleigh fading is used to model the channel. We assume that the channel matrix H is known, and stays constant for every 16 consecutively transmitted symbol vectors. The fading coefficients h_{ij}s of H are independent and identically-distributed (i.i.d.) complex unit variance Gaussian variables.

The energy per transmitted information bit is defined as $E_b = 1$, and the noise power is calculated from a given SNR $E_b/N_0|_{db}$ with $\sigma^2 = N_0$. There are $R_c M_c$ bits in a transmitted symbol, and the average symbol energy per transmit antenna is defined as $E_s = R_c M_c E_b$. Hence, the total energy per received antenna is $M_T E_s$. The MIMO detector calculates the soft-value information using the Max-log approximation for each received symbol vector. The LLR clipping values of ±8 are used for a prescribed bit when no counter-hypothesis bit is found in the candidate solution set [28]. Each LLR is a signed 10-bit long number with 6 bits in the fractional part. The Turbo decoder consists of two parallel concatenated soft-output Viterbi algorithm (SOVA) based decoders.

The Turbo decoder takes the soft outputs from the MIMO detector, and decodes the information bits iteratively. Eight iterations are run for each packet. For BER less than 10^{-2}, simulations are kept running until at least 1,000 bit errors are accumulated at the outputs. For BER greater than 10^{-2}, simulations are kept running until at least 100 bit errors are accumulated at the outputs.

5.1 BER Performance

Figure 4a shows the BER performance of LFSD, PFSD, LORD and K-best decoders for a 4×4 system with 16-QAM modulations. The LFSD algorithm is a soft-output extension of the FSD algorithm [10], where the number of expanded nodes doubles for p levels after the initial q FS levels. Hence, the cardinality vector $\underline{t}$ for 64 candidate LFSD decoder is [1, 2, 2, 16] with $q=1$ and $p=2$. We study 2 setups for the PFSD decoder, with 64 total candidates but different LLR computation complexities. The first uses the low complexity LLR algorithm described in Section 4.2. The other, denoted as the max-log LLR, computes LLR values at each bit position from all 64 candidate solution. Finally, the 64 candidate LORD algorithm [22] is added for baseline performance comparison.

The SNR differences of these algorithms are examined at 10^{-3} BER. The PFSD with parallel QR (pQR) and exact max-log LLR computation provides the best performance at 3.3 dB SNR. This is due to the

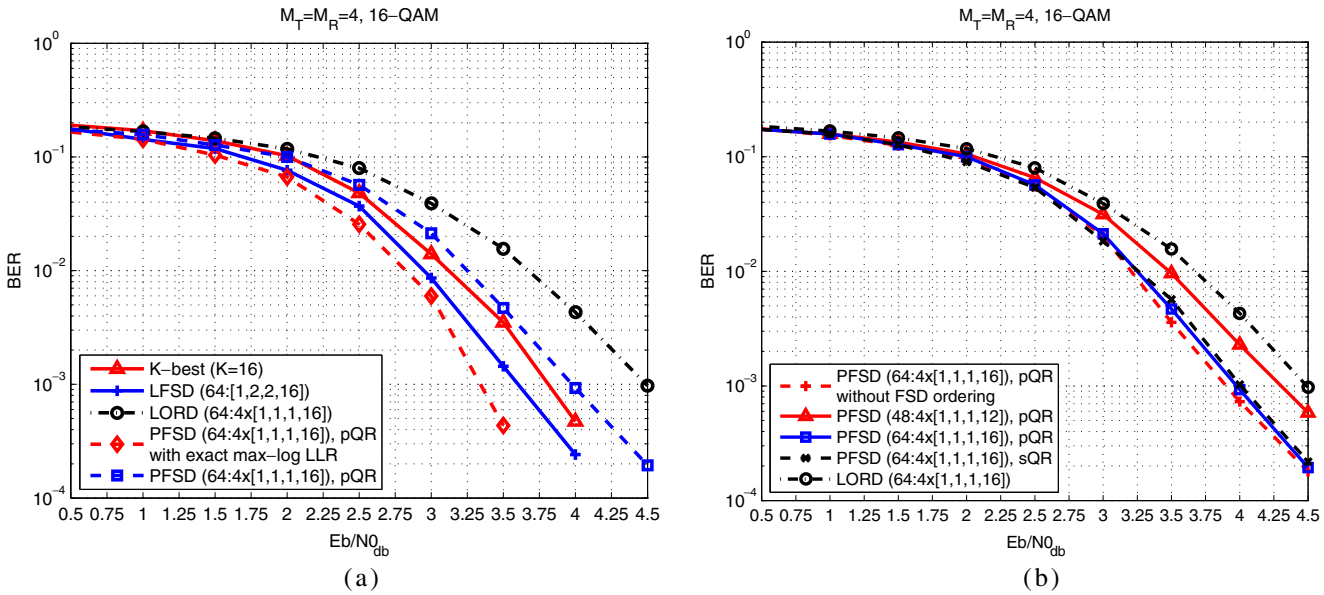


Figure 4 BER performance comparison of: **a** the K-best, the LFSD, the LORD and the PFSD algorithms and **b** the PFSD algorithm with different candidate cardinality vectors and QR decomposition setups with a rate $R_c = 1/2$ Turbo code and 16-QAM modulation.

existence of counter-hypothesis bits for all bit positions in their candidates solution set. The LFSD and the K-best decoders can not make such a claim. They use LLR clipping when necessary, which affects the quality of the soft bit values for the channel decoder, and in turn, the BER [29]. The LFSD and the K-best decoders achieve 10^{-3} BER at 3.6 dB and 3.8 dB, respectively. The PFSDs with low complexity LLR closely follow the K-best decoder. They achieve 10^{-3} BER slightly below 4dB. In low complexity LLR, the counter-hypothesis bit squared Euclidian distance is calculated from $\frac{1}{M_T}$ candidate vectors in the candidate set. This provides significant savings in terms of number of sorting operations at the expense of small BER degradation. The LORD algorithm also uses $\frac{1}{M_T}$ candidate lists to compute LLR values at all bit positions; however it does not find the exact quasi-ML solution. Hence, the overall quality of its LLR values is lower than PFSD with low complexity LLR.

Figure 4b shows the BER performances of the PFSD algorithm with low complexity LLR for different QR decomposition, initial channel ordering and cardinality vectors. We study 3 setups for PFSD with 64 candidates, (i) parallel QR (pQR) with initial channel ordering [16] for the first FSD, (ii) pQR without initial channel ordering, and (iii) serial QR (sQR), as shown in Fig. 2, for all M_T QR decompositions. For these three setups, the SNR differences are less than 0.1 dB. These results illustrate that initial channel ordering and QR decomposition scheme have little effect on the PFSD algorithm. Multiple FSDs reduce the likelihood that unfavorable channel ordering from any one FSD dominates the overall performance of PFSD. We conclude that PFSD can be implemented without performing the initial channel ordering to reduce preprocessing computation overhead.

Figure 4b also shows the effect of different number of nodes expanded at the top level of each FSD in PFSD. PFSD with 64 and 48 total candidates achieves 10^{-3} BER at 3.98 dB and 4.3 dB, respectively. PFSD with 48 candidate vectors provides better performance than LORD with 64 candidate vectors. The number of nodes expanded at the top level has significant impact, because less nodes expanded at the top layer increase the likelihood that the correct symbol will be missed at the top decoding layer of each FSD, and reduce the likelihood that symbols at lower layers will be detected correctly.

5.2 Computation Complexity

For high throughput applications, in addition to BER performance, the computation complexity as well as the overhead of parallelization are important metrics. Table 2 shows the operation costs of aforementioned algorithms measured in terms of number of additions, multiplications and comparisons. Among the candidate algorithms, LFSD (64:[1,2,2,16]) has the lowest number of multiplications and K-best has the highest number of multiplications. The proposed PFSD with 64 and 48 candidates has 1.3–1.6 times more multiplications than LFSD, but 10–33% less multiplications than the K-best.

Table 2 Comparison of SD algorithms with respect to performance and complexity.

Algorithms for 4 × 4 16-QAM system	SNR for 10^{-3} BER	Number of operations		
		Addition	Multiplication	Comparison
PFSD (64:4×[1,1,1,16])	3.98	3,456	1,680	135
PFSD (48:4×[1,1,1,12])	4.3	2,592	1,264	99
PFSD (64:4×[1,1,1,16]) with exact max-log	3.3	3,456	1,680	896
LORD (64:4×[1,1,1,16])	4.5	3,456	1,680	128
LFSD (64:[1,2,2,16])	3.6	2,080	1,008	825
K-best ($K = 16$)	3.8	3,024	1,872	4,199

Even though PFSD has more multiplications than LFSD, its candidate vectors are independent of each other and they can be generated in parallel. In contrast, in LFSD, the sibling candidate nodes are partially dependent on each other in intermediate decoding stages. This data dependency translates to additional computations [30], and increases decoder latency for larger antenna system with more intricate cardinality vector setup.

All these algorithms require comparisons for the sorting operations in solution set generation and LLR value calculation. The two PFSDs and LORD have similar number of comparisons for LLR value calculation, but LFSD and K-best have significantly higher number of comparisons. The PFSD implementation consists of M_T identical sorting networks and 1 quasi-ML path sorter. Each parallel sorting network requires at most $[2(\sqrt{M}-1)\sqrt{M}] + [M_c(\sqrt{M}-2)]$ comparators; the quasi-ML path sorter requires $2(M_T - 1)$ comparators. Hence, sorting networks for PFSD scale well with respect to increasing QAM size and antenna configuration. The sorting operations in LFSD and K-best depend on total number of candidate vectors and are typically implemented by a folded network which increases the latency of the LLR computation.

6 FPGA Implementation of PFSD

In this section, we discuss the hardware implementation details of the PFSD decoder for a 4 × 4 MIMO system. In the channel preprocessing part, we do not consider channel matrix ordering since it does not enhance the performance. The parallel QR architecture is based on Fig. 3b, where each P node is implemented with 6 real multipliers and 14 real adders, and each N node is implemented with 4 real multipliers, 2 real adders, 1 real divider and 1 square root. The outputs of two matrices come out every time unit starting with the 5th time unit, where the time unit is determined by the delay in the N node. We do not describe the QR implementation here but instead focus on the scalable parallel implementation of the PFSD parallel tree search and the PFSD low complexity LLR value calculation on an FPGA.

6.1 Architecture Details

The FPGA-based architecture for the PFSD algorithm with LLR calculation is shown in Fig. 5. 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 processes the data, and stores the results in the output memory. The SDRAM controller reads the results and store it back to the SDRAM. A 64-bit Xilinx Multi-Port Memory Controller (MPMC) [31] is used to implement the SDRAM controller, and a 128-bit Process Local Bus (PLB) [32] 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 $\underline{\hat{y}}_i$ to generate one LLR vector $\underline{l}$ (see Algorithm 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. The PLB transfers 128-bit each clock cycle. Hence, 4 pairs of R_i and $\underline{\hat{y}}_i$ require 12 read

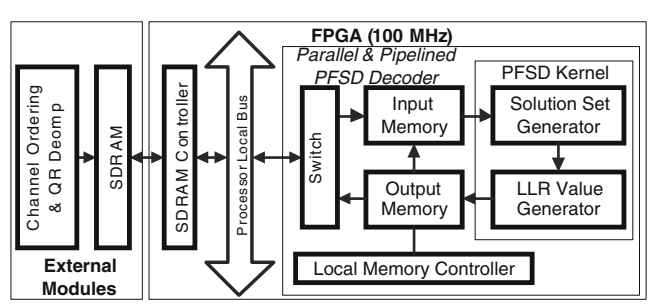


Figure 5 The proposed architecture of PFSD.

cycles, and an output vector $\underline{l}$ requires 2–4 write cycles depending on the QAM size. This bus can support 75 Mbps throughput for 4×4 64-QAM decoder running at 100 MHz. For higher throughput decoder configurations, wider PLB bus must be used to meet data I/O requirement.

Input/Output Memory There are 4 independent and identical dual-port input memory blocks. Each memory block is divided into three 128-bit wide 16-entry memory banks. One bank stores $\underline{\hat{y}}_i$, and other two banks each stores half of R_i. The single dual-port output memory block consists of six 128-bit wide 16-entry memory banks, where each entry of a memory bank stores 4 LLRs. Multiple banks in input and output memory blocks are used to ensure parallel data access for the PFSD kernel.

PFSD Kernel The block diagram of the PFSD kernel is shown in Fig. 6a. 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. The solution set generator has 4 parallel FSD cores. The ith core takes input data $\underline{\hat{y}}_i$ and R_i, and calculate path metrics of d ($d \leq M$) candidate solution vectors. Figure 6b shows the internal structure of a FSD core, which has two function units–sibling node enumeration and candidate path search. The sibling node enumeration finds d symbols expanded at the top level. It starts with a

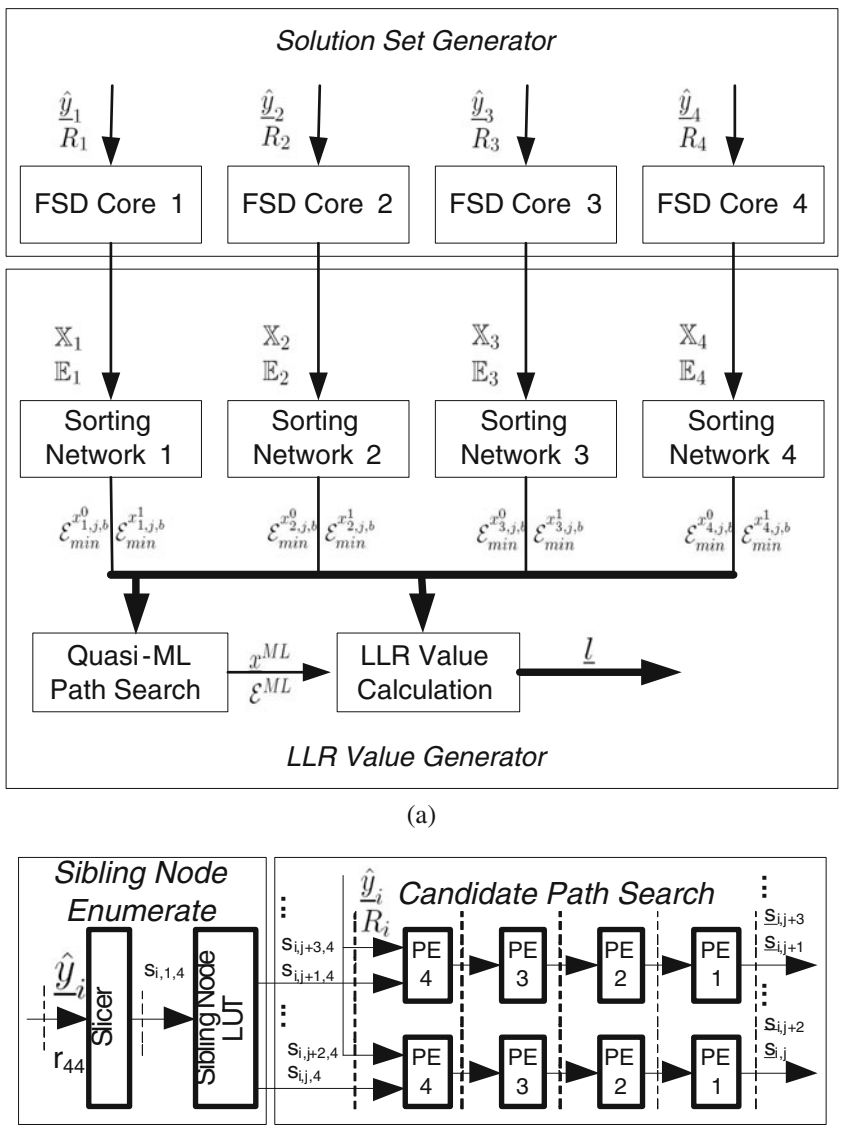


Figure 6 Block diagram of: **a** PFSD kernel and **b** FSD core.

slicer, which determines the top level 1st candidate symbol $s_{i,1,4}$. Remaining $d-1$ symbols are easily found by using a LUT table, where neighboring symbols of $s_{i,1,4}$ are listed in increasing order of their distance. The candidate path search unit takes 2 top level symbol inputs ($s_{i,j,4}$ and $s_{i,j+1,4}$, $1 \le j \le M$) and generates 2 candidate vector outputs ($\underline{s}_{i,j}$ and $\underline{s}_{i,j+1}$) per clock cycle. The candidate path search unit consists of two identical 4-stage pipeline connected processing elements (PE). PE1 to PE3 performs the following three tasks:

1. Calculating residual vector $\mathcal{V}_{i+1}$ according to Eq. 5, where only shift and add operations are used to obtain $r_{ij}s_j$.
2. Determine the 1st child symbol s_i by comparing the real and imaginary values of $\mathcal{V}_{i+1}$ to products of r_{ii} and all possible $\sqrt{M}$ PAM symbols.
3. Calculating PED $\mathcal{T}_i = \sum_{i=M_T}^{i} \mid \mathcal{D}_i(s_{(i)}) \mid^2$, and passing symbol vector and its PED to the next PE.

PE4 only needs to perform task 3.

Figure 6a shows that the LLR value generator consists of 4 sorting networks, a quasi-ML solution search module, and a LLR value calculation module. Outputs $\mathbb{X}_i$ and $\mathbb{E}_i$ from FSD core i are fed to sorting network i, which finds the minimum path metrics $\mathcal{E}_{\min}^{x_{i,b}^0}$ and $\mathcal{E}_{\min}^{x_{i,b}^1}$ for all M_c bit position of $s_{i,j,4}$, $1 \le j \le M$ (see Algorithm 3, lines 5 and 6). To accommodate parallel and pipelined sorting of 64-QAM path metrics, each sorting network has 12 parallel 2 stage sorters. The minimum path metrics from 4 sorting networks are passed to the LLR value calculation module and recorded in a metric table. There are 24 entries in the table, and each entry is 32-bit wide. The top 16 bits are for $\mathcal{E}_{\min}^{x_{i,b}^0}$ and the bottom 16 bits are for $\mathcal{E}_{\min}^{x_{i,b}^1}$. Minimum path metrics and their associated symbols from 4 sorting networks are passed to the quasi-ML solution search module. Once $\underline{s}^{ML}$ is found, the entries in the metric tables are updated. The LLR value calculation module then calculates the LLR vector $\underline{l}$ by using $\underline{l}^0$ and $\underline{l}^1$ from the metric table. There are 24 parallel subtractors in the LLR value calculation module. When operating under lower QAM configuration, such as 4-QAM and 16-QAM, the LLR value generator disables unused sorters in each sorting network, and asserts maximum path metric value for bit positions that do not exist in the metric table. A simple address counter is used to ensure that valid LLRs are passed to the output memory.

6.2 Virtex-4 FPGA Implementation

The VHDL code for the proposed PFSD architecture is developed in Xilinx ISE 10.1 environment. The RTL code is synthesized for the Xilinx Virtex-4 (X4VLX160) device with −12 speed grade. Table 3 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 solution set generator and the low complexity LLR value generator occupy 82.2% and 18.3% of the PFSD decoder slices. 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. In the Virtex-4 FPGA, we can fit two PFSDs provided that the multiplications are done by both DSP48s and LUTs. Each PFSD can be clocked at 120MHz. The critical path delay resides in the sorting network, where current path metrics are compared with existing path metrics.

The latency of the PFSD decoder is 55 cycles. Table 4 shows the maximum latency required for the individual modules. Each FSD core takes 45 cycles to generate one candidate vector for a new set of inputs, which includes the 22 cycles required by the FSD shared

Table 3 FPGA device utilization summary for PFSD and LLR calculation ($f = 100$ MHz, $N = 4$, $k = 2$).

Xilinx XC4VLX160		Device utilization				
		Slice flip flops	4 input LUTs	Slices	RAMB16	DSP48s
PFSD total area		32,464 (24%)	46,325 (34%)	25,787 (38%)	48 (16%)	64 (66%)
Solution set generator	FSD cores	26,280	38,060	20,256	12	64
	FSD shared	1,304	1,016	680		
LLR value generator	Sorting network	3,904	5,148	3,672		
	Quasi-ML search	352	375	271	4	
	LLR value calc	504	1,512	768		
Local memory controller		120	215	140		
Input memory					16	
Output memory					16	

Table 4 FPGA device timing table for PFSD and LLR calculation ($f = 100$ MHz, $N = 4$, $k = 2$).

Xilinx XC4VLX160	Solution set generator		LLR value generator		
	FSD cores	FSD shared	Sorting network	Quasi-ML search	LLR value calc
Clock cycle count	45	(22)	4	4	2

module. It contributes to 82% of the PFSD decoder latency. The latency of the parallel sorting network is 4 cycles. The quasi-ML search only adds 4 cycles to the overall decoder latency; its latency increases linearly with respect to M_T. Hence, full parallel implementation of the low complexity LLR algorithm greatly reduces overall decoder latency with moderate increase in decoder size.

For 4×4 64-QAM decoder running at 100 MHz, 24 LLRs are produced by the LLR value generator every 32 cycles. This translates to a throughput of 75 Mbps. However, when 16-QAM is used, 16 LLRs are produce by the LLR value generator every 14 cycles and the throughput is 114 Mbps instead of the expected 200 Mbps. This is because of the constraints imposed by the 128-bit wide PLB bus, where 14 read and write cycles are required (12 read cycles for inputs and 2 write cycles for outputs).

The power consumption of the PFSD implementation for a internal source voltage of 1.140 V is reported to be 4.36 W, of which the dynamic power consumption is 2.94 W, and the static power consumption is 1.42 W.

PFSD Scalability The decoding rate of the proposed PFSD decoder, R_d, depends on f, the circuit operating frequency, N, the number of FSD cores, d, the number of candidates generated by each core per received vector, and, k, the number of candidates generated by each core in each clock cycle for the case when the PLB bus width is wide enough to not pose a constraint. It is expressed by the following equation

$$R_d = f\,k\,N\frac{M_c}{d}, \text{where } 1 \le N \le M_T, 1 \le d, k \le M \qquad (9)$$

There are FPGA implementations of sphere decoders as early as 2006 [33]. The FSD decoder reduces computation complexity by using l^1 norm, generates 8 candidate vectors every clock cycle, and achieves 450 Mbps throughput for 64-QAM configuration when clocked at 150 MHz. The proposed soft-output PFSD generates up to 256 candidate vectors for 64-QAM configuration, uses l^2 norm instead of l^1 norm for higher decoder performance, and consequently has lower throughput. Recently, another FPGA implementation for hard-output FSD has been proposed in [34]. The decoder can achieve 52.5 Mbps at 35 MHz for 64-QAM system. Our PFSD implementation has larger area compared to [34] due to the FSD cores and the LLR value generator that are required for soft-output generation, along with use of larger bit width (16 vs 11). Also, the solution set generator in PFSD produces 8 candidate vector every clock cycle, whereas the decoder in [34] only produces 4 candidate vectors every clock cycle. In addition, l^1 norm is used in [34]. Most recently, an improved version of LORD [35] has been proposed and implemented in ASIC for a small system (2×2) with 64-QAM. It achieves a very high throughput of 240 Mbps with a 80 MHz clock. Since it is a smaller system, it allows high degree of parallelization where 32 candidate vectors are generated every clock cycle. Such a high degree of parallelization cannot be maintained for a 4×4 system with 64-QAM without a very large area overhead.

The parameter f is determined by the maximum clocking frequency of the FPGA implementation, and N is determined by the MIMO antenna configuration. The parameter d is determined by the sibling node enumeration unit within each FSD core. While R_d can be increased by decreasing d, it comes at the cost of lower BER performance, as demonstrated in Section 5, where 64-candidate PFSD outperforms 32-candidate PFSD. R_d can also be improved by increasing k. This solution does not degrade BER performance, but requires additional hardware resources.

7 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 performs better than LFSD and k-best in a 4×4 16-QAM system for configurations where all three algorithms have comparable computation complexity. The PFSD algorithm has been implemented on Xilinx VC4VLX160 FPGA. For 4×4 64-QAM configuration, the PFSD decoder

can achieve 75Mbps running at 100MHz. The scalability of the PFSD decoder is also investigated. Since the data paths of the PFSD decoder is inherently parallel, it can be easily mapped onto multiple FPGA chips to achieve very high decoding rate.

References

1. Butler, M. R. G., & Collings, I. B. (2004). A zero-forcing approximate log-likelihood receiver for MIMO bit-interleaved coded modulation. *IEEE Communication Letter, 8*, 105–107.
2. Foschini, G. J. (1996). Layered space-time architecture for wireless communication in a fading environment when using multi-element antenna. *Bell Labs Technical Journal, 1*, 41–59.
3. Mobasher, A., Taherzadeh, M., Sotirov, R., & Khandani, A. K. (2005). A near maximum likelihood decoding algorithm for MIMO systems based on semi-definite programming. In *IEEE international symposium on information theory (ISIT)* (pp. 1686–1690).
4. Sidiropoulos, N. D., & Luo, Z.-Q. (2006). A semidefinite relaxation approach to MIMO detection for high-order QAM constellations. *IEEE Signal Processing Letters, 13*, 525–528.
5. Pohst, M. (1981). On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. In *ACM special interest group on symbolic and algebraic manipulation (SIGSAM Bull.)* (Vol. 15, pp. 37–44).
6. Hassibi, B., & Vikalo, H. (2005). On the sphere-decoding algorithm I, expected complexity. *IEEE Transactions on Signal Processing, 53*, 2806–2818.
7. Guo, Z., Nilsson, P. (2006). Algorithm and implementation of the K-best sphere decoding for MIMO detection. *IEEE Transactions on Selected Areas in Communications, 44*, 491–503.
8. Chen, S., Zhang, T., & Xin, Y. (2007). Relaxed K-best MIMO signal detector design and VLSI implementation. *IEEE Transactions on VLSI Systems, 15*, 328–337.
9. Bhagawat, P., Dash, R., & Choi, G. (2009). Systolic like soft-detection architecture for 4×4 64-QAM MIMO system. In *IEEE The design, automation, and test in Europe (DATE)* (pp. 870–873).
10. Barbero, L. G., & Thompson, J. S. (2008). Extending a fixed-complexity sphere decoder to obtain likelihood information for Turbo-MIMO systems. *IEEE Transactions on Vehicular Technology, 57*, 2804–2814.
11. Caire, G., Taricco, G., & Biglieri, E. (1998). Bit-interleaved coded modulation. *IEEE Transaction of Information Theory, 8*, 927–946.
12. Studer, C., Burg, A., & Bölcskei, H. (2008). Soft-output sphere decoding: Algorithms and VLSI implementation. *IEEE Transactions on Selected Areas in Communications, 26*, 290–300.
13. Fincke, U., & Pohst, M. (1985). Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation, 44*, 161–163.
14. Schnorr, C. P., & Euchner, M. (1991). Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Fundamentals of Computation Theory, 529*, 68–85.
15. Wong, K. W., Tsui, C. Y., Cheng, R. S.-K., & Mow, W. H. (2002). A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels. In *IEEE international symposium on circuits and systems (ISCAS)* (Vol. 3, pp. 273–276).
16. Barbero, L. G., & Thompson, J. S. (2006). A fixed-complexity MIMO detector based on the complex sphere decoder. In *IEEE international workshop on signal processing advances for wireless communications (SPAWC)* (pp. 1–5).
17. Hess, C., Wenk, M., Burg, A., Luethi, P., Studer, C., Felber, N., et al. (2007). Reduced-complexity MIMO detector with close-to-ML error rate performance. In *ACM Great Lakes symposium on VLSI* (pp. 200–203).
18. Li, M., Bougard, B., Lopez, E. E., Bourdoux, A., Novo, D., Van Der Perre, L., et al. (2008). Selective spanning with fast enumeration: A near maximum-likelihood MIMO detector designed for parallel programmable baseband architectures. In *IEEE international conference on communications (ICC)* (pp. 737–741).
19. Milliner, D. L., Zimmermann, E., Barry, J. R., & Fettweis, G. P. (2008). A framework for fixed complexity breadth-first MIMO detection. In *IEEE international symposium on spread spectrum techniques and applications (ISSSTA)* (pp. 129–132).
20. Li, M., Novo, D., Bougard, B., Naessens, F., Van der Perre, L., & Catthoor, F. (2008). An implementation friendly low complexity multiplierless LLR generator for soft MIMO sphere decoders. In *IEEE workshop on signal processing systems (SiPS)* (pp. 118–123).
21. Siti, M., & Fitz, M. P. (2007). On layer ordering techniques for near-optimal MIMO detectors. In *IEEE wireless communications and networking conference (WCNC)* (pp. 1199–1204).
22. Siti, M., & Fitz, M. P. (2006). A novel soft-output layered orthogonal lattice detector for multiple antenna communications. In *IEEE international conference on communications (ICC)* (pp. 1686–1691).
23. Jalden, J., Barbero, L. G., Ottersten, B., & Thompson, J. S. (2009). The error probability of the fixed-complexity sphere decoder. *IEEE Transactions on Signal Processing, 57*, 2711–2720.
24. El-Amawy, A., & Dharmarajan, K. R. (1989). Parallel VLSI algorithm for stable inversion of dense matrices. In *Computers and digital techniques, IEE proceedings E* (Vol. 236, pp. 575–580).
25. Liu, K. R., Hsieh, S.-F., & Yao, K. (1992). Systolic block householder transformation for RLS algorithm with two-level pipelined implementation. *IEEE Transactions on Signal Processing, 40*, 946–958.
26. Wübben, D., Böhnke, R., Rinas, J., Kühn, V., & Kammeyer, K. D. (2001). Efficient algorithm for decoding layered space-time codes. *IEEE Transactions on Electronics Letters, 37*, 1348–1350.
27. Vucetic, B., & Yuan, J. (2000). *Turbo codes: Principles and applications*. Norwell: Kluwer.
28. Hochwald, B. M., & Brink, S. (2003). Achieving near-capacity on a multiple-antenna channel. *IEEE Transactions on Communications, 51*, 389–399.
29. Milliner, D. L., Zimmermann, E., Barry, J. R., & Fettweis, G. (2008). Channel state information based LLR clipping in list MIMO detection. In *IEEE international symposium on personal, indoor and mobile radio communications (PIMRC)* (pp. 1–5).
30. Burg, A., Borgmann, M., Wenk, M., Zellweger, M., Fichtner, W., & Bolcskei, H. (2005). VLSI implementation of MIMO detection using the sphere decoding algorithm. *IEEE Journal of Solid-State Circuits, 40*, 1566–1577.
31. Xilinx. Xilinx multi-port memory controller. http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf.
32. Xilinx. Processor local bus. http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.
33. Barbero, L. G., & Thompson, J. S. (2006). *FPGA design considerations in the implementation of a fixed-throughput sphere decoder for MIMO systems*. In *IEEE international workshop*

on signal processing advances for wireless communications (SPAWC) (pp. 1–5).
34. Bhagawat, P., Dash, R., & Choi, G. (2008). Architecture for reconfigurable MIMO detector and its FPGA implementation. In *IEEE international conference on electronics, circuits and systems (ICECS)* (pp. 61–64).
35. Cupaiuolo, T., Siti, M., & Tomasoni, A. (2010). Low-complexity high throughput VLSI architecture of soft-output ML MIMO detector. In *Design, automation test in Europe conference exhibition (DATE), 2010* (pp. 1396–1401).

Qi Qi received the B.S., M.S. and Ph.D. degrees in electrical engineering from Arizona State University (ASU), Tempe, in 2001, 2004 and 2010, respectively. His research interests include VLSI architectures and algorithms for communication and signal processing systems.

Chaitali Chakrabarti (SM'02) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1984, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, in 1986 and 1990, respectively. She is a Professor with the Department of Electrical Engineering, Arizona State University (ASU), Tempe. Her research interests include the areas of low power embedded systems design including memory optimization, high level synthesis and compilation, and VLSI architectures and algorithms for signal processing, image processing, and communications. Prof. Chakrabarti was a recipient of the Research Initiation Award from the National Science Foundation in 1993, a Best Teacher Award from the College of Engineering and Applied Sciences from ASU in 1994, and the Outstanding Educator Award from the IEEE Phoenix Section in 2001. She served as the Technical Committee Chair of the DISPS subcommittee, IEEE Signal Processing Society (2006-2007). She is now an Associate Editor of the Journal of VLSI Signal Processing Systems and the IEEE Transactions on Very Large Scale Integration Systems.