

Efficient Square-root and Division Free Algorithms for Inverse LDL^T Factorization and the Wide-sense Givens Rotation with Application to V-BLAST

Hufei Zhu

Huawei Technology Co., Ltd.

Wen Chen

Shanghai Jiaotong Univ.

Bin Li

Huawei Technology

Abstract—We propose efficient square-root and division free algorithms for inverse LDL^T factorization and the wide-sense Givens rotation, mainly for conventional multiply-add digital computers. The algorithms are then employed to propose a fast algorithm for vertical Bell Laboratories Layered Space-Time architecture (V-BLAST). The proposed inverse LDL^T factorization avoids the conventional back substitution of the Cholesky factor, and then has the speedups of 1.19 over the square root and division free alternative Cholesky factorization plus the back substitution. The proposed wide-sense Givens rotation requires the same computational complexity as the efficient Givens rotation. The speedups of the proposed V-BLAST algorithm over the recursive V-BLAST algorithm are 1.05 ~ 1.4. The difference between the complexity of the proposed V-BLAST algorithm and that of the square-root V-BLAST algorithm is trivial and even negligible. However, the square-root V-BLAST algorithm requires many square-root and division operations, while the proposed V-BLAST algorithm requires no square-root operations, and requires only one division for each transmit signal, which can be postponed until the signal estimation is required.

I. INTRODUCTION

A practical version of Bell Laboratories Layered Space-Time architecture (BLAST), i.e. vertical BLAST (V-BLAST), can achieve very high spectral efficiency in rich multi-path environments [1]. However, it requires high computational complexity. Thus fast algorithms have been proposed for V-BLAST [2]–[5], of which typical examples are the square-root algorithm in [2] and the recursive algorithm in [3]. Recently the improved square-root and recursive algorithms with reduced complexity were proposed in [4] and [5], respectively.

On the other hand, the square-root V-BLAST algorithm usually requires many Givens rotations [2] and may adopt Cholesky factorization [6]. However, in their natural form, both the Givens rotation and Cholesky factorization require a relatively large number of square-root and division operations [6], [7], which are time-consuming computations for conventional multiply-add digital computers [8]. In fixed-point implementations, it is highly desirable to avoid square-root and division operations, since they are demanding in terms of the required bit precision and clock cycles [6], [7].

A specially configured computer, called a CORDIC, can be employed to avoid the square-root and division operations [6], [8], e.g., those in the Givens rotation [8]. For conventional multiply-add digital computers, the alternative Cholesky

factorization in [6] and the Givens-rotation-based algorithms in [7] can avoid both square-root and division operations. The alternative Cholesky factorization may be adapted to implement the square-root V-BLAST detector in [2], to reduce the required square-root and division operations [6]. However, it cannot offer a full square-root and division free algorithm for V-BLAST, since it can only be applied to implement a part of the *initialization* phase [6]. Furthermore, it requires more real multiplications than the conventional Cholesky factorization [6]. On the other hand, although the Givens-rotation-based algorithms can be treated as wide-sense rotations [7], they are no longer rotations since they do not possess the normalization property [7]. Moreover, they were proposed to substitute for the Givens rotations to perform QR decomposition [7]. Thus the Givens-rotation-based algorithms in [7] are not suitable to replace the Givens rotations in the square-root V-BLAST algorithm that is based on Cholesky factorization [6].

In this paper, we propose an efficient square-root and division free algorithm for inverse LDL^T factorization [9], which requires less computational complexity, with respect to the similar algorithm based on the square-root and division free Cholesky factorization in [6]. Then the proposed inverse LDL^T -factorization algorithm is applied to propose a V-BLAST algorithm, which includes a square-root and division free Givens-rotation-based algorithm proposed by us. The proposed V-BLAST algorithm is square-root free and nearly division free, and requires less complexity and divisions than the recursive V-BLAST algorithm in [5].

The V-BLAST System model is overviewed in section II. Then in section III, we propose the V-BLAST algorithm, including the square-root and division free algorithms for inverse LDL^T factorization and the wide-sense Givens rotation. In section IV, we derive the proposed algorithm for the inverse LDL^T factorization. The complexity of the presented algorithms is evaluated in Section V. Finally, we make conclusion in Section VI.

In what follows, $(\bullet)^{-1}$, $(\bullet)^T$, $(\bullet)^*$, and $(\bullet)^H$ denote matrix inversion, matrix transposition, matrix conjugate, and matrix conjugate transposition, respectively. \mathbf{I}_k is the identity matrix with size k , while $\mathbf{0}_k$ is the zero column vector of length k .

II. SYSTEM MODEL

The considered V-BLAST system consists of K transmit antennas and $N(\geq K)$ receive antennas in a rich-scattering and flat-fading wireless channel. At the transmitter, the data stream is de-multiplexed into K sub-streams. Then each sub-stream is encoded and fed to its respective transmit antenna. Let $\mathbf{s} = [s_1, s_2, \dots, s_K]^T$ denote the vector of transmit symbols from K antennas, and assume $E(\mathbf{ss}^H) = \sigma_s^2 \mathbf{I}_K$. Then the received symbol vector is

$$\mathbf{x} = \mathbf{H} \cdot \mathbf{s} + \mathbf{n}, \quad (1)$$

where \mathbf{n} is the $N \times 1$ complex Gaussian noise vector with zero mean and covariance $\sigma_n^2 \mathbf{I}_N$, and $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_K]$ is the $N \times K$ complex channel matrix with statistically independent entries. The vector \mathbf{h}_m denotes the m^{th} column of \mathbf{H} .

The minimum mean-square error (MMSE) detection of \mathbf{s} is

$$\hat{\mathbf{s}} = (\mathbf{H}^H \mathbf{H} + \alpha \mathbf{I}_K)^{-1} \mathbf{H}^H \mathbf{x}, \quad (2)$$

where $\alpha = \sigma_n^2 / \sigma_s^2$. Let

$$\mathbf{R} = \mathbf{H}^H \cdot \mathbf{H} + \alpha \mathbf{I}_K. \quad (3)$$

Then the estimation error covariance matrix is [2]

$$\mathbf{Q} = \mathbf{R}^{-1} = (\mathbf{H}^H \mathbf{H} + \alpha \mathbf{I}_K)^{-1}. \quad (4)$$

The conventional V-BLAST detects K entries of \mathbf{s} by K iterations with the optimal ordering. In each iteration, the entry with the highest post detection signal-to-noise ratio (SNR) among all the undetected entries is detected by an linear MMSE or zero-forcing (ZF) filter. Then its effect is substracted from the received symbol vector [1]–[3].

The first k columns of \mathbf{H} can be represented as

$$\mathbf{H}_k = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_k]. \quad (5)$$

Then \mathbf{R}_k and \mathbf{Q}_k are defined from \mathbf{H}_k by (3) and (4), respectively, while \mathbf{R}_k satisfies [3]

$$\mathbf{R}_k = \begin{bmatrix} \mathbf{R}_{k-1} & \mathbf{v}_{k-1} \\ \mathbf{v}_{k-1}^H & \beta_k \end{bmatrix}. \quad (6)$$

The recursive V-BLAST algorithm [5] utilizes \mathbf{Q}_k s ($k = K, K-1, \dots, 1$) to form the MMSE filters.

III. THE PROPOSED ALGORITHM FOR V-BLAST

We propose an efficient algorithm for V-BLAST in this section, which is square-root free and nearly division free. Instead of \mathbf{Q}_k utilized in [5], we use the LDL^T factorization of $\hat{\mathbf{Q}}_k$ to form the MMSE filter. Correspondingly we propose an efficient square-root and division free algorithm for inverse LDL^T factorization, which will be derived in section IV.

We use alternative LDL^T factors of \mathbf{R}_{k+1}^{-1} , i.e.,

$$\mathbf{L}_{k+1} (\mathbf{D}_{k+1} / \delta_{k+1}) \mathbf{L}_{k+1}^H = \mathbf{R}_{k+1}^{-1}. \quad (7)$$

In (7), \mathbf{L} , \mathbf{D} , and δ can be computed iteratively by

$$\left\{ \begin{array}{l} \mathbf{L}_{k+1} = \begin{bmatrix} \mathbf{L}_k & \mathbf{a}_k \\ \mathbf{0}_k^H & l_{k+1} \end{bmatrix}, \\ \mathbf{D}_{k+1} = \begin{bmatrix} \eta_{k+1} \mathbf{D}_k & \mathbf{0}_k \\ \mathbf{0}_k^H & d_{k+1} \end{bmatrix}, \end{array} \right. \quad (8a)$$

$$(8b)$$

and

$$\delta_{k+1} = \delta_k \eta_{k+1}, \quad (9)$$

where

$$\mathbf{a}_k = -\mathbf{L}_k \mathbf{D}_k \mathbf{L}_k^H \mathbf{v}_k, \quad (10)$$

and

$$\left\{ \begin{array}{l} l_{k+1} = \delta_k, \\ d_{k+1} = 1, \end{array} \right. \quad (11a)$$

$$\eta_{k+1} = \delta_k \beta_{k+1} - \mathbf{v}_k^H \mathbf{L}_k \mathbf{D}_k \mathbf{L}_k^H \mathbf{v}_k. \quad (11b)$$

$$(11c)$$

The iterations start from the initial

$$\left\{ \begin{array}{l} \mathbf{L}_1 = 1, \\ \mathbf{D}_1 = 1, \end{array} \right. \quad (12a)$$

$$\delta_1 = \mathbf{R}_1 = r_{1,1}. \quad (12b)$$

$$(12c)$$

In (8) and (11), δ_k , \mathbf{L}_k and \mathbf{D}_k also satisfy (7).

The above-described iterations will lead to numerically unlimited results, which may cause a problem in fixed-point implementations [6]. We can alleviate this problem by scaling, as in [6]. Scaling is achieved by dividing (or multiplying) only by powers of 2 [6], which is a shift operation in binary fixed-point implementation. Since δ_k is complex, we can keep $|\delta_k|^2$ between 0.25 and 4, and scale the diagonal entries in \mathbf{D}_k accordingly. Thus δ_k and \mathbf{D}_k in (12) or in (8) and (9) are multiplied by c_k , which is a power of 2. Correspondingly in each iteration we end up with $\delta_k c_k$ and $\mathbf{D}_k c_k$, while $|\delta_k c_k|^2$ is always between 0.25 and 4.

Now let us propose the step 1 for the *initialization* phase, which includes the following sub-steps.

The Sub-steps of Step 1 for Initialization

- 1-a) Assume the successive detection order to be t_K, t_{K-1}, \dots, t_1 . Correspondingly permute \mathbf{H} to be $\mathbf{H} = \mathbf{H}_K = [\mathbf{h}_{t_1}, \mathbf{h}_{t_2}, \dots, \mathbf{h}_{t_K}]$, and permute \mathbf{s} to be $\mathbf{s} = \mathbf{s}_K = [s_{t_1}, s_{t_2}, \dots, s_{t_K}]^T$.
- 1-b) Utilize the permuted \mathbf{H} to compute \mathbf{R}_K , where we can obtain all \mathbf{R}_{k-1} s, \mathbf{v}_{k-1} s and β_k s (for $k = K, K-1, \dots, 2$) directly [3], as shown in (6).
- 1-c) Compute \mathbf{L}_k , \mathbf{D}_k and δ_k from \mathbf{L}_{k-1} , \mathbf{D}_{k-1} and δ_{k-1} iteratively, by (8a), (8b), (9), (10) and (11). The initial $\mathbf{L}_{|K|} = \mathbf{L}_K$, $\mathbf{D}_{|K|} = \mathbf{D}_K$ and $\delta = \delta_K$ is obtained after $K-1$ iterations, which start from \mathbf{L}_1 , \mathbf{D}_1 and δ_1 in (12).
- 1-d) Compute $\mathbf{z}_K = \mathbf{H}_K^H \mathbf{x}$.

In the conventional algorithm [10] performing matrix inversion by Cholesky factorization, the inverse Cholesky factor is computed from the Cholesky factor by the back substitution (for triangular matrix inversion), an inherent serial process unsuitable for the parallel implementation [11]. For example, we can choose the square-root and division free alternative Cholesky factorization [6] to factorize \mathbf{R} into $\mathbf{R} = \mathbf{W}^H \mathbf{D}^{-1} \mathbf{W}$ and obtain

$$\mathbf{Q} = \mathbf{R}^{-1} = \mathbf{W}^{-1} \mathbf{D} \mathbf{W}^{-H}, \quad (13)$$

where \mathbf{W} is triangular, and \mathbf{D}^{-1} is the inverse of the diagonal \mathbf{D} . Since usually K divisions are required in (13) to compute \mathbf{W}^{-1} from the $K \times K$ triangular \mathbf{W} by the back substitution [10], the full algorithm is not division free. However, the proposed algorithm for inverse LDL^T factorization is more efficient (as will be shown in section V) and division free, since it computes the inverse Cholesky factor of \mathbf{R} from \mathbf{R} directly, and does not need the back substitution of the Cholesky factor.

From the LDL^T factorization of \mathbf{Q} , we can propose an efficient algorithm for V-BLAST as follows, which is square-root free and nearly division free.

The Proposed V-BLAST Algorithm

Initialization:

- 1) Set $k = K$. Compute the initial \mathbf{R}_K , \mathbf{z}_K , $\mathbf{L}_{|K} = \mathbf{L}_K$, $\mathbf{D}_{|K} = \mathbf{D}_K$ and $\delta = \delta_K$. This step includes the above-described sub-steps 1-a, 1-b, 1-c and 1-d.

Iterative Detection:

- 2) Find the minimum diagonal entry in

$$\bar{\mathbf{Q}}_{|k} = \delta \mathbf{Q}_{|k} = \mathbf{L}_{|k} \mathbf{D}_{|k} \mathbf{L}_{|k}^H, \quad (14)$$

which is assumed to be the i^{th} one. Then permute the i^{th} row of $\mathbf{L}_{|k}$ to be the last (k^{th}) row. Permute \mathbf{s} , \mathbf{z}_k , rows and columns in \mathbf{R}_k accordingly [5].

- 3) Find a non-unitary transformation Θ and the corresponding diagonal $\mathbf{D}'_{|k}$ that satisfy

$$\bar{\mathbf{Q}}_{|k} = (\mathbf{L}_{|k} \Theta) \mathbf{D}'_{|k} (\mathbf{L}_{|k} \Theta)^H = \mathbf{L}_{|k} \mathbf{D}'_{|k} \mathbf{L}_{|k}^H, \quad (15)$$

while Θ block upper-triangularizes $\mathbf{L}_{|k}$, i.e.,

$$\mathbf{L}_{|k} \Theta = \begin{bmatrix} \mathbf{L}_{|k-1} & \mathbf{u}_{k-1} \\ \mathbf{0}_{k-1}^T & \lambda_k \end{bmatrix}, \quad (16)$$

where \mathbf{u}_{k-1} and λ_k are a column vector and a scalar, respectively. Then let $\mathbf{L}_{|k} = \mathbf{L}_{|k} \Theta$ and $\mathbf{D}_{|k} = \mathbf{D}'_{|k}$.

- 4) Use the last row in $\mathbf{Q}_{|k} = \mathbf{L}_{|k} \mathbf{D}_{|k} \mathbf{L}_{|k}^H / \delta$, to form the MMSE estimate [5] by

$$\hat{s}_k = \frac{1}{\delta} \left[\begin{bmatrix} \mathbf{0}_{k-1}^T & \lambda_k \end{bmatrix} \mathbf{D}_k \begin{bmatrix} \mathbf{L}_{k-1} & \mathbf{u}_{k-1} \\ \mathbf{0}_{k-1}^T & \lambda_k \end{bmatrix} \right]^H \mathbf{z}_k \\ = (1/\delta) \lambda_k d_k \left[\begin{bmatrix} \mathbf{u}_{k-1}^H & \lambda_k^* \end{bmatrix} \mathbf{z}_k \right]. \quad (17)$$

- 5) Obtain s_k from \hat{s}_k via slicing.
- 6) Cancel the effect of s_k in \mathbf{z}_k by

$$\mathbf{z}_{k-1} = \mathbf{z}_k^{[-1]} - s_k \cdot \mathbf{v}_{k-1}, \quad (18)$$

where $\mathbf{z}_k^{[-1]}$ is \mathbf{z}_k with the last entry removed, and \mathbf{v}_{k-1} is in \mathbf{R}_k [5], as shown in (6).

- 7) If $k > 1$, go back to step 2 with $\mathbf{L}_{|k-1}$, $\mathbf{D}_{|k-1}$, \mathbf{R}_{k-1} and \mathbf{z}_{k-1} , instead of $\mathbf{L}_{|k}$, $\mathbf{D}_{|k}$, \mathbf{R}_k and \mathbf{z}_k , respectively. Correspondingly let $k = k-1$. Notice that $\mathbf{L}_{|k-1}$, $\mathbf{D}_{|k-1}$ and \mathbf{R}_{k-1} are obtained by removing the last row and column in $\mathbf{L}_{|k}$, $\mathbf{D}_{|k}$ and \mathbf{R}_k , respectively.

In step 3, the transformation Θ can be performed by a series of wide-sense Givens rotations. To avoid divisions, represent d_j in $\mathbf{D}_{|k}$ by a numerator and a denominator, i.e.,

$$d_j = \bar{d}_j / \xi_j, \quad (19)$$

while the initial $\bar{d}_j = d_j$ and $\xi_j = 1$. Then we form $\Psi_k^{j,j+1}$ and $\mathbf{D}_k'^{j,j+1}$. $\Psi_k^{j,j+1}$ is equal to \mathbf{I}_k except the 2×2 sub-block in the j^{th} and $(j+1)^{th}$ rows and columns, which is

$$\begin{bmatrix} \psi_{j,j}^{k,j} & \psi_{j,j+1}^{k,j} \\ \psi_{j+1,j}^{k,j} & \psi_{j+1,j+1}^{k,j} \end{bmatrix} = \begin{bmatrix} |l_k^{j+1}|^2 & (l_k^{j+1})^* l_k^{j+1} \bar{d}_j \xi_{j+1} \\ -(l_k^{j+1})^* l_k^{j+1} & |l_k^{j+1}|^2 \bar{d}_{j+1} \xi_j \end{bmatrix}, \quad (20)$$

where l_k^i denotes the i^{th} entry in the last k^{th} row of \mathbf{L}_k . The corresponding $\mathbf{D}_k'^{j,j+1}$ is equal to $\mathbf{D}_{|k}$ except the j^{th} and $(j+1)^{th}$ diagonal entries, i.e. d'_j and d'_{j+1} . Use (19) to represent d'_j and d'_{j+1} as \bar{d}'_j / ξ'_j and $\bar{d}'_{j+1} / \xi'_{j+1}$, respectively. Then these numerators and denominators are computed by

$$\left\{ \begin{array}{l} \bar{d}'_j = \bar{d}_j \bar{d}_{j+1}, \\ \xi'_j = \psi_{j,j}^{k,j} \psi_{j+1,j+1}^{k,j} - \psi_{j,j+1}^{k,j} \psi_{j+1,j}^{k,j}, \end{array} \right. \quad (21a)$$

$$\left\{ \begin{array}{l} \bar{d}'_{j+1} = 1, \\ \xi'_{j+1} = \xi'_j \xi_j \xi_{j+1}. \end{array} \right. \quad (21b)$$

$$\left\{ \begin{array}{l} \bar{d}'_{j+1} = 1, \\ \xi'_{j+1} = \xi'_j \xi_j \xi_{j+1}. \end{array} \right. \quad (21c)$$

$$\left\{ \begin{array}{l} \bar{d}'_{j+1} = 1, \\ \xi'_{j+1} = \xi'_j \xi_j \xi_{j+1}. \end{array} \right. \quad (21d)$$

From the efficient complex Givens rotation [12], we derive $\Psi_k^{j,j+1}$ and the corresponding $\mathbf{D}_k'^{j,j+1}$, which are computed by (20) and (21), respectively. It is easy to verify $\Psi_k^{j,j+1} \mathbf{D}_k'^{j,j+1} (\Psi_k^{j,j+1})^H = \mathbf{D}_{|k}$. Then we can conclude that $\mathbf{L}_{|k} \Psi_k^{j,j+1}$ and $\mathbf{D}_k'^{j,j+1}$ satisfy (15) (where $\Theta = \Psi_k^{j,j+1}$). Moreover, it can be seen that $\Psi_k^{j,j+1}$ transforms the j^{th} and $(j+1)^{th}$ entries in each row of $\mathbf{L}_{|k}$, and zeros the j^{th} entry in the last k^{th} row.

Now the non-unitary transformation Θ can be

$$\Theta = \Psi_k^{i,i+1} \Psi_k^{i+1,i+2} \dots \Psi_k^{k-1,k} = \prod_{j=i}^{k-1} \Psi_k^{j,j+1}, \quad (22)$$

while after each $\Psi_k^{j,j+1}$, $\mathbf{D}_{|k}$ must be updated to be $\mathbf{D}_k'^{j,j+1}$ accordingly. Since $\mathbf{L}_{|K}$ is upper triangular, we can apply the Θ in (22) when $k = K$. In step 2, we can delete the i^{th} row in $\mathbf{L}_{|K}$ firstly to get $\underline{\mathbf{L}}_{|K}$, and then add the deleted i^{th} row to $\underline{\mathbf{L}}_{|K}$ as the last row to obtain the permuted $\mathbf{L}_{|K}$. Then it is easy to verify that the $\mathbf{L}_{|K-1}$ obtained from $\mathbf{L}_{|K} \Theta$ by (16) is still upper triangular. Thus we can conclude that all $\mathbf{L}_{|k}$ s ($k = K, K-1, \dots, 2$) in step 3 are upper triangular, i.e., the Θ in (22) can be applied to each $\mathbf{L}_{|k}$.

We can also perform scaling in (20) and (21), as in the inverse LDL^T -factorization algorithm and [6]. Let c_i ($i = 1, 2, 3$) denote a power of 2. We can simply multiply \bar{d}'_j and ξ'_j by c_1 to keep $\bar{d}'_j c_1$ between 0.25 and 4. Then $\xi'_j c_1$ and the first column of (20) are multiplied by $(c_2)^2$ and c_2 , respectively, to keep $\xi'_j c_1 (c_2)^2$ between 0.25 and 4. On the other hand, ξ'_{j+1} and the second column of (20) are multiplied by $(c_3)^2$ and c_3 , respectively, to keep $\xi'_{j+1} (c_3)^2$ between 0.25 and 4.

IV. THE DERIVATION OF THE SQUARE-ROOT AND DIVISION FREE INVERSE LDL^T FACTORIZATION

In this section, we will derive (7), (8), (9), (10) and (11). The LDL^T factorization [9] of \mathbf{R} can be represented as

$$\mathbf{L}_\mathbf{R} \mathbf{D}_\mathbf{R} \mathbf{L}_\mathbf{R}^H = \mathbf{R}, \quad (23)$$

where $\mathbf{L}_\mathbf{R}$ is unit lower triangular, and $\mathbf{D}_\mathbf{R}$ is diagonal [9]. From (23), we can obtain

$$\mathbf{L}_\mathbf{R}^{-H} \mathbf{D}_\mathbf{R}^{-1} \mathbf{L}_\mathbf{R}^{-1} = \mathbf{R}^{-1}, \quad (24)$$

where $\mathbf{L}_\mathbf{R}^{-1}$, the inverse of the unit lower triangular matrices $\mathbf{L}_\mathbf{R}$, must be unit lower triangular [9]. Define $\tilde{\mathbf{D}} = \mathbf{D}_\mathbf{R}^{-1}$ and the unit upper triangular $\tilde{\mathbf{L}} = \mathbf{L}_\mathbf{R}^{-H}$. Then (24) becomes

$$\tilde{\mathbf{L}} \tilde{\mathbf{D}} \tilde{\mathbf{L}}^H = \mathbf{R}^{-1}, \quad (25)$$

which is an alternative LDL^T factorization of \mathbf{R}^{-1} .

Represent \mathbf{R} as \mathbf{R}_{K+1} when it is a $(K+1) \times (K+1)$ matrix. Correspondingly represent the LDL^T factorization of \mathbf{R}_{K+1} as $\hat{\mathbf{L}}_{K+1}$, $\hat{\mathbf{D}}_{K+1}$ and $\hat{\mathbf{M}}_{K+1}$. To reuse the previously listed equations for brevity, we use (\hat{i}) to express the equation (i) with each variable χ replaced by $\hat{\chi}$. Then we can represent \mathbf{R}_{K+1} , $\hat{\mathbf{L}}_{K+1}$ and $\hat{\mathbf{D}}_{K+1}$ by (6), (8a) and (8b), respectively, where $\hat{l}_{K+1} = \hat{\eta}_{K+1} = 1$. From (8a) and (8b), we deduce

$$\left\{ \begin{array}{l} \hat{\mathbf{L}}_{K+1}^{-1} = \begin{bmatrix} \hat{\mathbf{L}}_K^{-1} & -\hat{\mathbf{L}}_K^{-1} \hat{\mathbf{a}}_K \\ \mathbf{0}_K^H & 1 \end{bmatrix}, \\ \hat{\mathbf{D}}_{K+1}^{-1} = \begin{bmatrix} \hat{\mathbf{D}}_K^{-1} & \mathbf{0}_K \\ \mathbf{0}_K^H & 1/\hat{d}_{K+1} \end{bmatrix}. \end{array} \right. \quad (26a)$$

$$\left\{ \begin{array}{l} \hat{\mathbf{D}}_{K+1}^{-1} = \begin{bmatrix} \hat{\mathbf{D}}_K^{-1} & \mathbf{0}_K \\ \mathbf{0}_K^H & 1/\hat{d}_{K+1} \end{bmatrix}. \end{array} \right. \quad (26b)$$

From (25), we can obtain

$$\hat{\mathbf{L}}_{K+1}^{-H} \hat{\mathbf{D}}_{K+1}^{-1} \hat{\mathbf{L}}_{K+1}^{-1} = \mathbf{R}_{K+1}. \quad (27)$$

Then substitute (26) into (27) to obtain

$$\left[\begin{array}{c} \hat{\mathbf{L}}_K^{-H} \hat{\mathbf{D}}_K^{-1} \hat{\mathbf{L}}_K^{-1} \\ -\hat{\mathbf{a}}_K^H \hat{\mathbf{L}}_K^{-H} \hat{\mathbf{D}}_K^{-1} \hat{\mathbf{L}}_K^{-1} \end{array} \right] = \left[\begin{array}{c} -\hat{\mathbf{L}}_K^{-H} \hat{\mathbf{D}}_K^{-1} \hat{\mathbf{L}}_K^{-1} \hat{\mathbf{a}}_K \\ \hat{\mathbf{a}}_K^H \hat{\mathbf{L}}_K^{-H} \hat{\mathbf{D}}_K^{-1} \hat{\mathbf{L}}_K^{-1} \hat{\mathbf{a}}_K + \hat{d}_{K+1}^{-1} \end{array} \right]$$

$$= \left[\begin{array}{c} \mathbf{R}_K & \mathbf{v}_K \\ \mathbf{v}_K^H & \beta_{K+1} \end{array} \right], \text{ i.e., }$$

$$\left\{ \begin{array}{l} \hat{\mathbf{L}}_K^{-H} \hat{\mathbf{D}}_K^{-1} \hat{\mathbf{L}}_K^{-1} = \mathbf{R}_K, \end{array} \right. \quad (28a)$$

$$\left\{ \begin{array}{l} -\hat{\mathbf{L}}_K^{-H} \hat{\mathbf{D}}_K^{-1} \hat{\mathbf{L}}_K^{-1} \hat{\mathbf{a}}_K = \mathbf{v}_K, \end{array} \right. \quad (28b)$$

$$\left\{ \begin{array}{l} \hat{\mathbf{a}}_K^H \hat{\mathbf{L}}_K^{-H} \hat{\mathbf{D}}_K^{-1} \hat{\mathbf{L}}_K^{-1} \hat{\mathbf{a}}_K + \hat{d}_{K+1}^{-1} = \beta_{K+1}. \end{array} \right. \quad (28c)$$

From (28b), we can derive that $\hat{\mathbf{a}}_K$ is computed by (10) where $\hat{\mathbf{v}}_k = \mathbf{v}_k$. Substitute (10) into (28c) to deduce

$$\hat{d}_{K+1} = 1 / (\beta_{K+1} - \mathbf{v}_K^H \hat{\mathbf{L}}_K \hat{\mathbf{D}}_K \hat{\mathbf{L}}_K^H \mathbf{v}_K). \quad (29)$$

From (28a), we can conclude that the submatrices $\hat{\mathbf{L}}_k$, $\hat{\mathbf{D}}_k$ and \mathbf{R}_k ($K \geq k \geq 1$) in $\hat{\mathbf{L}}$, $\hat{\mathbf{D}}$ and \mathbf{R} also satisfy (27) and (25). From (25), we obtain the initial $\hat{\mathbf{L}}_1 = 1$ and $\hat{\mathbf{D}}_1 = 1/\mathbf{R}_1 = 1/r_{1,1}$. Then we can compute $\hat{\mathbf{L}}_{k+1}$ and $\hat{\mathbf{D}}_{k+1}$ from $\hat{\mathbf{L}}_k$ and $\hat{\mathbf{D}}_k$ iteratively by (10) and (8) for $k = 1, 2, \dots, K$, where $\hat{l}_{k+1} = \hat{\eta}_{k+1} = 1$, $\hat{\mathbf{v}}_k = \mathbf{v}_k$, and \hat{d}_{K+1} is computed by (29).

It requires divisions to compute (29) and $\hat{\mathbf{D}}_1 = 1/r_{1,1}$. To avoid divisions, we employ (12) to define \mathbf{L}_1 , \mathbf{D}_1 and δ_1 ,

which satisfy (7). Assume that (7) is satisfied for any k ($k = 1, 2, \dots, K+1$), which is substituted into (25) to obtain

$$\hat{\mathbf{L}}_k \hat{\mathbf{D}}_k \hat{\mathbf{L}}_k^H = \mathbf{L}_k (\mathbf{D}_k / \delta_k) \mathbf{L}_k^H. \quad (30)$$

It can be seen from (30) that \mathbf{L}_k and \mathbf{D}_k / δ_k are equivalent to $\hat{\mathbf{L}}_k$ and $\hat{\mathbf{D}}_k$, respectively. Thus we can form $\tilde{\mathbf{L}}_{k+1}$ and $\tilde{\mathbf{D}}_{k+1}$ by (8), i.e. (8) with each variable χ replaced by $\tilde{\chi}$, where $\tilde{\mathbf{L}}_k = \mathbf{L}_k$, $\tilde{\mathbf{D}}_k = \mathbf{D}_k / \delta_k$, $\tilde{\mathbf{a}}_k = \hat{\mathbf{a}}_k$, $\tilde{d}_{k+1} = \hat{d}_{k+1}$, and $\tilde{l}_{k+1} = \hat{\eta}_{k+1} = 1$. Substitute (8) into $\tilde{\mathbf{L}}_{k+1} \tilde{\mathbf{D}}_{k+1} \tilde{\mathbf{L}}_{k+1}^H$. Then from (30) and (25), we can verify

$$\tilde{\mathbf{L}}_{k+1} \tilde{\mathbf{D}}_{k+1} \tilde{\mathbf{L}}_{k+1}^H = \hat{\mathbf{L}}_{k+1} \hat{\mathbf{D}}_{k+1} \hat{\mathbf{L}}_{k+1}^H = \mathbf{R}_{k+1}^{-1}. \quad (31)$$

Substitute (30) into (10), which is then substituted into $\tilde{\mathbf{a}}_k = \hat{\mathbf{a}}_k$ to represent $\tilde{\mathbf{a}}_k$ by $\mathbf{L}_k (\mathbf{D}_k / \delta_k) \mathbf{L}_k^H$. Correspondingly in (8a), represent $\tilde{\mathbf{a}}_k$ by $\mathbf{L}_k (\mathbf{D}_k / \delta_k) \mathbf{L}_k^H$ to obtain

$$\tilde{\mathbf{L}}_{k+1} = \begin{bmatrix} \tilde{\mathbf{L}}_k & \tilde{\mathbf{a}}_k \\ \mathbf{0}_k^H & \tilde{l}_{k+1} \end{bmatrix} = \mathbf{L}_{k+1} \begin{bmatrix} \mathbf{I}_k & \mathbf{0}_k \\ \mathbf{0}_k^H & \frac{1}{\delta_k} \end{bmatrix}. \quad (32)$$

In (32), \mathbf{L}_{k+1} is defined by (8a), where l_{k+1} and \mathbf{a}_k are defined in (11a) and (10). Thus we have derived (8a), (10) and (11a).

From (32), it can be seen that to make

$$\mathbf{L}_{k+1} \frac{\mathbf{D}_{k+1}}{\delta_{k+1}} \mathbf{L}_{k+1}^H = \tilde{\mathbf{L}}_{k+1} \tilde{\mathbf{D}}_{k+1} \tilde{\mathbf{L}}_{k+1}^H, \quad (33)$$

we only need to let

$$\frac{\mathbf{D}_{k+1}}{\delta_{k+1}} = \begin{bmatrix} \mathbf{I}_k & \mathbf{0}_k \\ \mathbf{0}_k^H & \frac{1}{\delta_k} \end{bmatrix} \tilde{\mathbf{D}}_{k+1} \begin{bmatrix} \mathbf{I}_k & \mathbf{0}_k \\ \mathbf{0}_k^H & \frac{1}{\delta_k^*} \end{bmatrix}^H. \quad (34)$$

Then from (33) and (31), we can conclude that \mathbf{L}_{k+1} , \mathbf{D}_{k+1} and δ_{k+1} satisfy (7).

To simplify (34), substitute (30) into (29), which can replace \hat{d}_{k+1} in (8b) since $\hat{d}_{k+1} = \hat{d}_{k+1}$. Moreover, $\tilde{\mathbf{D}}_k = \mathbf{D}_k / \delta_k$. So $\tilde{\mathbf{D}}_{k+1}$ in (8b) can be represented by \mathbf{L}_k , \mathbf{D}_k and δ_k , which is substituted into (34) to simplify it to

$$\frac{\mathbf{D}_{k+1}}{\delta_{k+1}} = \begin{bmatrix} \frac{\mathbf{D}_k}{\delta_k} & \mathbf{0}_k \\ \mathbf{0}_k^H & \frac{1}{\delta_k \eta_k} \end{bmatrix} = \frac{1}{\delta_k \eta_k} \begin{bmatrix} \eta_k \mathbf{D}_k & \mathbf{0}_k \\ \mathbf{0}_k^H & 1 \end{bmatrix}, \quad (35)$$

where η_k satisfies (11c). From (35), we can derive (9), (8b) and (11b).

Now we have derived (8), (9), (10) and (11). In the above discussion, we have assumed that (7) is satisfied for any k ($k = 1, 2, \dots, K+1$), which needs to be proved. We have verified that if the assumption (7) is satisfied for any k , it will also be satisfied for the corresponding $k+1$. It can be seen from (12) that the assumption (7) is satisfied for $k=1$. Thus we can conclude that the assumption (7) is satisfied for all subsequent $(k+1)$ s, where $k = 1, 2, \dots, K$.

V. COMPLEXITY EVALUATION

In sub-step 1-c, we compute $\mathbf{g}_k = \mathbf{L}_k^H \mathbf{v}_k$ firstly. Then we compute (10) and (11c) by $\mathbf{a}_k = -\mathbf{L}_k \mathbf{D}_k \mathbf{g}_k$ and $\eta_{k+1} = \delta_k \beta_{k+1} + \mathbf{v}_k^H \mathbf{a}_k$, respectively, where \mathbf{L}_k is always triangular. In step 3, $\Psi_k^{j,j+1}$ is computed by (20), which includes 2 real value and 2 complex value. Then $\Psi_k^{j,j+1}$ only requires 3 complex multiplications and 1 complex additions equivalently

[3] to transform a row, while it only transforms non-zero entries in the first $j + 1$ rows of the triangular $\mathbf{L}_{|k|}$.

As in [4], we assume the equal probabilities for $i = 1, 2, \dots, k$ in step 2 to compute the average complexity, while in sub-step 1-a we assume the detection order to be the optimal order of the last frame in quasi-stationary channels or of the adjacent subcarrier in MIMO OFDM systems, to reduce the average complexity of step 3, which can even be zero.

We count a square-root operation or a real division as one floating-point operation (flop) [3]. To reduce the complexity, the square-root V-BLAST algorithm in [4] can adopt the efficient Givens rotation [12] with 2 square-root operations and 2 real divisions, which requires the same number of flops as the proposed wide-sense Givens rotation (i.e. (20) and (21)). Then compared to the V-BLAST algorithm in [4], the proposed square-root and division free V-BLAST algorithm only requires $\frac{1}{2}K^2$ more of real multiplications for $\eta_{k+1}\mathbf{D}_k$ in (8b). The real multiplications for $\eta_{k+1}\mathbf{D}_k$ in (8b) and those by $\hat{\eta}_{k+1}$ in (10) can be avoided by using (10) and (8) with \hat{d}_{K+1} computed by (29) and $\hat{l}_{k+1} = \hat{\eta}_{k+1} = 1$. The corresponding V-BLAST algorithm increases K real divisions, and requires $\frac{5}{2}K^2 - \frac{1}{2}K^2 = 2K^2$ less of flops than that in [4].

Let (j, k) denote the complexity of j complex multiplications and k complex additions, and simplify (j, k) to (j) if $j = k$. To compute the inverse Cholesky factor of \mathbf{R} , the alternative Cholesky factorization in [6] plus the back substitution (for matrix inversion) [9], [10] requires $\frac{1}{2}K^3 + O(K^2)$ more of real multiplications [6] (i.e. the complexity increase of the algorithm in [6] with respect to the conventional Cholesky factorization) than the proposed inverse LDL^T algorithm with a complexity of $(\frac{1}{3}K^3)$. Then the speedups of the latter over the former in the number of flops are 1.19.

The complexity of the proposed V-BLAST algorithm ranges from $(\frac{2}{3}K^3 + \frac{1}{2}K^2N, \frac{4}{9}K^3 + \frac{1}{2}K^2N)$ to $(\frac{1}{3}K^3 + \frac{1}{2}K^2N)$, while that of the recursive V-BLAST algorithm in [5] is $(\frac{2}{3}K^3 + \frac{1}{2}K^2N)$. Thus the speedups of the former over the latter in the number of flops are $1.05 \sim 1.4$. Moreover, the former needs K divisions, while the latter needs $2K$ divisions [5].

Assume $K = N$. For different number of transmit/receive antennas, we carried out some numerical experiments to count the average flops of the presented algorithms. The results are shown in Fig. 1. It can be seen that they are consistent with the theoretical flops calculation.

VI. CONCLUSION

Mainly for conventional multiply-add digital computers, we propose the inverse LDL^T factorization [9] and the wide-sense Givens rotation that both are square-root and division free. Then they are applied to propose a fast algorithm for V-BLAST. The proposed inverse LDL^T factorization avoids the conventional back substitution (for triangular matrix inversion) of the Cholesky factor, and then has the speedups of 1.19 over the square-root and division free alternative Cholesky factorization [6] plus the back substitution with K divisions. The proposed wide-sense Givens rotation requires the same computational complexity as the efficient Givens rotation [12].

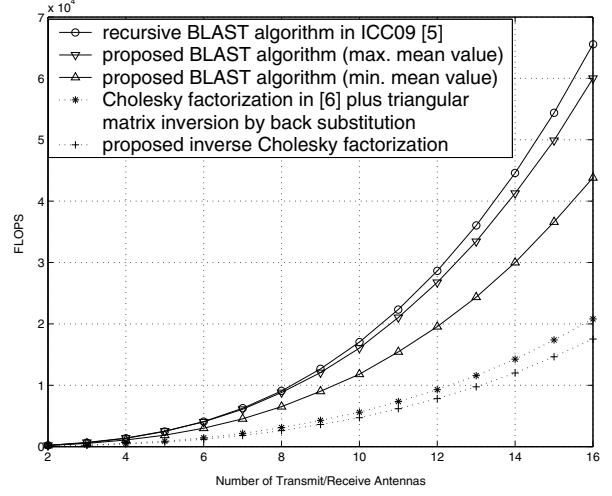


Fig. 1. Complexity Comparison among the Presented Algorithms.

The complexity difference between the proposed V-BLAST algorithm and the square-root V-BLAST algorithm in [4] is only $O(K^2)$, and then can be neglected. However, the algorithm in [4] requires many (averagely about $K(K+1)/2$) square-root and division operations, while the proposed V-BLAST algorithm is square-root free, and requires only one real division for each transmit signal, which can be postponed until the signal estimation is required. With respect to the recursive V-BLAST algorithm in [5], the proposed V-BLAST algorithm has the speedups of $1.05 \sim 1.4$ and requires only half divisions.

REFERENCES

- [1] P. W. Wolniansky, G. J. Foschini, G. D. Golden and R. A. Valenzuela, "V-BLAST: an architecture for realizing very high data rates over the rich-scattering wireless channel", *Proc. ISSE 98*, pp. 295-300, 1998.
- [2] B. Hassibi, "An efficient square-root algorithm for BLAST", *IEEE ICASSP '00*, pp. 737-740, June 2000.
- [3] J. Benesty, Y. Huang and J. Chen, "A fast recursive algorithm for optimum sequential signal detection in a BLAST system", *IEEE Trans. on Signal Processing*, pp. 1722-1730, July 2003.
- [4] H. Zhu, W. Chen, D. Chen, Y. Du and J. Lu, "Reducing the Computational Complexity for BLAST by Using a Novel Fast Algorithm to Compute an Initial Square-root Matrix," *IEEE VTC 2008 Fall*.
- [5] H. Zhu, W. Chen and F. She, "Improved Fast Recursive Algorithms for V-BLAST and G-STBC with Novel Efficient Matrix Inversion," *IEEE ICC 2009*, Dresden, Germany, June 2009.
- [6] L. M. Davis, "Scaled and decoupled Cholesky and QR decompositions with application to spherical MIMO detection", *IEEE WCNC*, 2003.
- [7] E. N. Frantzeskakis and K. J. R. Liu, "A class of square root and division free algorithms and architectures for QRD-based adaptive signal processing", *IEEE Trans. on Signal Processing*, Sep 1994.
- [8] J. G. Proakis, C. M. Rader, F. Ling, C. L. Nikias, M. Moonen and I. K. Proudler, *Algorithms for Statistical Signal Processing*, Prentice Hall 2002.
- [9] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.
- [10] A. Burian, J. Takala, M. Ylinen, "A fixed-point implementation of matrix inversion using Cholesky decomposition", *IEEE International Symposium on MHS*, Dec. 2003, Vol. 3, pp. 1431-1434.
- [11] E. J. Baranowski, "Triangular factorization of inverse data covariance matrices", *IEEE ICASSP 91*, Apr. 1991.
- [12] D. Bindel, J. Demmel, W. Kahan and O. Marques, "On Computing Givens rotations reliably and efficiently", *ACM Transactions on Mathematical Software (TOMS) archive*, Vol. 28, Issue 2, June 2002.