High-Throughput Lattice Reduction for Large-Scale MIMO Systems based on Seysen's Algorithm

Farhana Sheikh, Alexios Balatsoukas-Stimming, and Chia-Hsiang Chen
Intel Labs, Hillsboro, OR, USA

***Abstract*—Lattice reduction aided MIMO detection has attracted significant attention recently due to its low complexity and excellent error rate performance. The most popular lattice reduction algorithms are the Lenstra-Lenstra-Lovász algorithm and Seysen's algorithm, although the former has received much more attention than the latter. In this work, we present a simplification to Seysen's lattice reduction algorithm, which reduces the per-iteration computational complexity from quadratic to linear with a small degradation in the quality of the resulting reduced lattice. Moreover, we present an efficient VLSI architecture which demonstrates the advantages of the proposed algorithm and can achieve a throughput of up to 91 Mmatrices/s at an operating frequency of 1 GHz.**

I. Introduction

In a multiple-input and multiple output (MIMO) system multiple antennas are used both at the transmitter and at the receiver in order to increase its spectral efficiency. MIMO technology has been embraced by several recent communications standards, such as IEEE 802.11ac (Wi-Fi) and 3GPP Long Term Evolution (4G), where relatively small arrays of up to four antennas are used. MIMO is expected to continue playing a major role in emerging 5G systems, where it may be employed on a much more massive scale.

At the receiver side, MIMO detection can be performed using various algorithms. Maximum-likelihood detection is optimal with respect to the error rate, but its complexity, which is exponential in the number of antennas, is prohibitive for practical implementations. Linear detection algorithms, such as zero-forcing (ZF) detection and minimum mean square error (MMSE) detection, have low complexity but their performance is degraded significantly with respect to the ML detector. The performance gap to the ML detector can be reduced by using significantly more complex non-linear algorithms, which are usually based on successive interference cancellation (SIC). While SIC-based non-linear detectors improve the diversity gain at low-to-medium SNRs, they still show a diversity loss at higher SNR values.

Lattice reduction (LR) enables MIMO detection with maximum-likelihood (ML) diversity order using low-complexity linear detection algorithms [1]. The main idea behind LR-aided MIMO detection is to transform the lattice basis formed by the MIMO channel matrix into a "more orthogonal" basis, perform MIMO detection on this new basis, and finally transform the result back to the original basis. While exact LR has exponential complexity, there exist several approximate (and mostly iterative) LR algorithms, which exhibit both low complexity and excellent LR performance. In the context of MIMO detection, the most popular LR algorithm is the Lenstra-Lenstra-Lovász algorithm (LLL) [2], which has received significant attention in theoretical studies (e.g., [1], [3]) as well as in VLSI implementations (e.g., [4], [5], [6], [7]). An alternative and promising LR algorithm is Seysen's algorithm (SA) [8], which, although known to have better error rate performance when paired with linear MIMO detectors (see, e.g., [9]), has attracted much less interest than the LLL algorithm. Specifically, some algorithmic simplifications were considered in [10], [11], while only a small number of VLSI implementations has been presented [12], [13].

Contribution: In this work, we present a reduced-complexity SA-based algorithm, called *simplified SA* (SSA). The main advantage of this algorithm is that the quadratic per-iteration complexity of the original SA is reduced to linear with a small degradation in the quality of the reduced lattice, thus making SSA scale more favorably to future large-scale MIMO systems than both the original SA and the LLL algorithm. Moreover, we present a VLSI implementation of the proposed algorithm and a comparison with existing VLSI implementations of SA.

Outline: The remainder of this paper is organized as follows. In Section II, we provide some background on LR-aided MIMO detection and Seysen's algorithm. In Section III, we describe SSA and we compare its performance with the performance of the LLL algorithm and the original SA. In Section IV we briefly describe a VLSI implementation of SSA and we compare it with existing VLSI implementations, Finally, Section V concludes this paper.

II. LR-Aided MIMO Detection

In this section, we provide background on LR-aided MIMO detection as well as on Seysen's algorithm and on some existing hardware-oriented simplifications of Seysen's algorithm.

A. System Model and MIMO Detection

A MIMO system with N transmit and M receive antennas and with $M \geq N$ can be modeled as

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

where $\mathbf{H}$ denotes the $M \times N$ complex channel matrix, $\mathbf{s}$ denotes the complex transmitted symbol vector, $\mathbf{n}$ denotes the complex additive noise, and $\mathbf{y}$ denotes the complex received noisy symbol vector. We assume that $H_{i,j} \sim \mathcal{CN}(0,1), \forall i = 1,\dots,M,\ j = 1,\dots,N$, and $\mathbf{n} \sim \mathcal{CN}\left(0, \sigma^2\mathbf{I}\right)$. Moreover, for the symbol vector we have $\mathbf{s} \in \mathcal{M}^N$, where $\mathcal{M}$ is a set of constellation points. For example, for 4-QAM we have $\mathcal{M} = \{+1+1j, +1-1j, -1-1j, -1+1j\}$. The signal-to-noise ratio (SNR) at each antenna of the receiver is defined as $\text{SNR} = \frac{N\mathbb{E}\left(\|\mathbf{s}\|^2\right)}{\sigma^2}$.

The goal of any MIMO detector is to produce an estimate of $\mathbf{s}$, denoted by $\hat{\mathbf{s}}$, that is as accurate as possible. Under the aforementioned channel and noise model, the optimal ML detector can be written as

$$\hat{\mathbf{s}} = \arg\min_{\mathbf{s}\in\mathcal{M}^N} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2. \tag{2}$$

Linear detectors, on the other hand, can be written as a simple matrix product followed by an element-wise quantization operator as

$$\hat{\mathbf{s}} = \mathcal{Q}_{\mathcal{M}}\left(\mathbf{W}\mathbf{y}\right), \tag{3}$$

where $\mathbf{W}$ is a matrix that represents a linear filter operation and $\mathcal{Q}_{\mathcal{M}}$ simply quantizes each element of $\mathbf{W}\mathbf{y}$ to the nearest constellation point in $\mathcal{M}$. In a ZF and an MMSE detector, $\mathbf{W}$ is calculated as

$$\mathbf{W}^{\text{ZF}} = \left(\mathbf{H}^H\mathbf{H}\right)^{-1}\mathbf{H}^H, \tag{4}$$

$$\mathbf{W}^{\text{MMSE}} = \left(\mathbf{H}^H\mathbf{H} + N\sigma^2\mathbf{I}_N\right)^{-1}\mathbf{H}^H, \tag{5}$$

respectively, where $\mathbf{I}_N$ denotes the $N \times N$ identity matrix.

B. LR-Aided MIMO Detection

If we substitute the constellation $\mathcal{M}$ with $\mathcal{M}_{\text{r}} = \{\alpha + \beta j\}, \alpha, \beta \in \mathbb{Z}$, then the matrix $\mathbf{H}$ can be viewed as a lattice basis. The idea behind LR-aided linear MIMO detection is to transform $\mathbf{H}$ into a "more orthogonal" basis through a linear transformation $\mathbf{H}_{\text{r}} = \mathbf{T}\mathbf{H}$, where $\mathbf{T}$ is a complex unimodular matrix, and to perform simple linear detection on the relaxed constellation $\mathcal{M}_{\text{r}}$. Finally, the result is transformed back to the original lattice basis $\mathbf{H}$ and the original constellation $\mathcal{M}$.

The received vector needs to be scaled and shifted in order for the constellation points in $\mathcal{M}$ to correspond to complex numbers with consecutive integer-valued real and imaginary parts. For rectangular QAM constellations this can be achieved by the following transformation

$$\mathbf{y}_{\text{r}} = \frac{1}{2}\left(\mathbf{y} - \mathbf{H}(\mathbf{1} + \mathbf{1}j)_N\right), \tag{6}$$

where $(\mathbf{1} + \mathbf{1}j)_N$ denotes the $N \times 1$ all-ones complex vector. Linear detection on the reduced basis becomes

$$\hat{\mathbf{s}}_{\text{r}} = \lfloor \mathbf{W}_{\text{r}}\mathbf{y}_{\text{r}} \rceil, \tag{7}$$

where $\lfloor\cdot\rceil$ denotes the element-wise rounding operator and the linear filter $\mathbf{W}_{\text{r}}$ for ZF and MMSE detection can be calculated by replacing $\mathbf{H}$ with $\mathbf{H}_{\text{r}} = \mathbf{T}\mathbf{H}$ in (4) and (5). We can now get $\hat{\mathbf{s}}$ as

$$\hat{\mathbf{s}} = \mathcal{Q}_{\mathcal{M}}\left(2\mathbf{T}\hat{\mathbf{s}}_{\text{r}} + (\mathbf{1} + \mathbf{1}j)_N\right). \tag{8}$$

C. Seysen's Algorithm

The most difficult and computationally intensive part of LR-aided detection is the determination of the transformation matrix $\mathbf{T}$. While the computation of the optimal transformation matrix $\mathbf{T}$ has exponential complexity, there exist several approximate algorithms that provide lattice reduction results that are more than adequate for MIMO detection. In the context of MIMO detection, the most popular LR algorithms are the LLL algorithm and Seysen's algorithm. In this work, we focus on Seysen's algorithm [8], which has been demonstrated to provide better performance when paired with linear MIMO detectors [9].

Seysen's algorithm is a gradient descent algorithm that greedily minimizes Seysen's metric, which is defined as

$$S(\mathbf{H}) = \sum_{n=1}^{N} \|\mathbf{h}_n\|^2 \|\mathbf{h}_n^{-1}\|^2, \tag{9}$$

where $\mathbf{h}_n$ and $\mathbf{h}_n^{-1}$ denote the n-th column of $\mathbf{H}$ and $\mathbf{H}^{-1}$, respectively. Seysen's algorithm is initialized as $\mathbf{G} = \mathbf{H}^H\mathbf{H}$ and $\mathbf{T} = \mathbf{I}_N$ and performs multiple iterations of the following steps.

Step 1. λ coefficient calculation for $i, j = 1, \ldots, N,\ i \neq j$,

$$\lambda_{i,j} = \left\lfloor \frac{1}{2}\left(\frac{G_{j,i}^{-1}}{G_{i,i}^{-1}} - \frac{G_{j,i}}{G_{j,j}}\right)\right\rceil. \tag{10}$$

Step 2. Δ calculation (i.e., Seysen's metric reduction) for $i, j = 1, \ldots, N,\ i \neq j$,

$$\Delta_{i,j} = -2\left(G_{j,i}G_{i,i}^{-1}|\lambda_{i,j}|^2 - G_{j,j}\Re(\lambda_{i,j}^* G_{j,i}^{-1})\right. \tag{11}$$

$$\left. + \, G_{i,i}^{-1}\Re(\lambda_{i,j}^* G_{j,i})\right). \tag{12}$$

Step 3. Maximum Δ determination

$$\{s, t\} = \arg\max_{\substack{i,j=1,\ldots,N,\\ i\neq j}} \Delta_{i,j}. \tag{13}$$

Step 4. Matrix updates. More specifically, we have the $\mathbf{G}$ and $\mathbf{G}^{-1}$ matrix updates

$$G_{s,j}^{'} = G_{s,j} + \lambda_{s,j}^* G_{t,j}, \quad G_{j,s}^{'} = \left(G_{s,j}^{'}\right)^*,\ j \neq s, \tag{14}$$

$$G_{s,s}^{'} = G_{s,s} + 2\Re(\lambda_{s,t}^* G_{t,s}) + |\lambda_{s,t}|^2 G_{t,t}, \tag{15}$$

$$G_{t,j}^{-1'} = G_{t,j}^{-1} - \lambda_{t,j}G_{s,j}^{-1}, \quad G_{j,t}^{-1'} = \left(G_{t,j}^{-1'}\right)^*,\ j \neq t, \tag{16}$$

$$G_{t,t}^{-1'} = G_{t,t}^{-1} - 2\Re(\lambda_{s,t}G_{s,t}^{-1}) + |\lambda_{s,t}|^2 G_{s,s}^{-1}, \tag{17}$$

the optional[1] $\mathbf{H}$ matrix update

$$H_{j,s}^{\prime} = H_{j,s} + \lambda_{s,t}H_{j,t},\ j = 1, \ldots, M, \tag{18}$$

and the $\mathbf{T}$ and $\mathbf{T}^{-1}$ matrix updates

$$T_{j,s}^{\prime} = T_{j,s} + \lambda_{s,t}T_{j,t},\ j = 1, \ldots, M, \tag{19}$$

$$T_{t,j}^{-1'} = T_{t,j}^{-1} - \lambda_{s,t}T_{s,j}^{-1},\ j = 1, \ldots, M. \tag{20}$$

The algorithm terminates either when all $\lambda_{i,j} = 0$, or when a pre-determined maximum number of iterations is reached. An alternative early-termination method based on $\Delta_{i,j}$ was suggested in [11].

[1]This is only required if the detector that follows the LR step requires access to the reduced channel matrix, as in, e.g., SIC-based non-linear detectors that require the calculation of the QR decomposition of $\mathbf{H}_r$.

D. Existing Simplifications of SA

In [11] it was proposed to restrict the $\lambda_{i,j}$ coefficients to $\lambda_{i,j} \in \{\alpha + \beta j\},\ \alpha, \beta \in \{-1, 0, +1\}$. While this approximation seems very simple, it actually significantly reduces the complexity of SA because $\lambda_{i,j}$ can be computed without the need for division, which is very costly to implement in hardware. More specifically, let

$$d_{i,j} = G_{j,i}^{-1} G_{j,j} - G_{j,i} G_{i,i}^{-1}, \quad p_{i,j} = G_{i,i}^{-1} G_{j,j}. \tag{21}$$

Then, the real and imaginary parts of $\lambda_{i,j}$ can be calculated as

$$\Re(\lambda_{i,j}) = \begin{cases} 0, & |\Re(d_{i,j})| < p_{i,j}, \\ \text{sign}\left(\Re(d_{i,j})\right) \text{sign}\left(p_{i,j}\right), & \text{otherwise.} \end{cases} \tag{22}$$

$$\Im(\lambda_{i,j}) = \begin{cases} 0, & |\Im(d_{i,j})| < p_{i,j}, \\ \text{sign}\left(\Im(d_{i,j})\right) \text{sign}\left(p_{i,j}\right), & \text{otherwise.} \end{cases} \tag{23}$$

Moreover, several complex multiplications in all steps can be replaced by conditional additions. For example, $x \triangleq \lambda^* G$ in (14) (we drop the indices for simplicity) can be rewritten as

$$\Re(x) = \begin{cases} \Re(\lambda)\Re(G), & \Re(\lambda) \neq 0, \Im(\lambda) = 0, \\ \Im(\lambda)\Im(G), & \Re(\lambda) = 0, \Im(\lambda) \neq 0, \\ \Re(\lambda)\Re(G) + \Im(\lambda)\Im(G), & \Re(\lambda) \neq 0, \Im(\lambda) \neq 0, \\ 0, & \Re(\lambda) = 0, \Im(\lambda) = 0. \end{cases} \tag{24}$$

$$\Im(x) = \begin{cases} \Re(\lambda)\Im(G), & \Re(\lambda) \neq 0, \Im(\lambda) = 0, \\ -\Im(\lambda)\Re(G), & \Re(\lambda) = 0, \Im(\lambda) \neq 0, \\ \Re(\lambda)\Im(G) - \Im(\lambda)\Re(G), & \Re(\lambda) \neq 0, \Im(\lambda) \neq 0, \\ 0, & \Re(\lambda) = 0, \Im(\lambda) = 0. \end{cases} \tag{25}$$

In the same work, a K-Seysen algorithm was proposed, where the maximum Δ search is only performed over the first $K \leq K_{\max} \triangleq N(N-1)$ Δ values. We call SA with the aforementioned approximations Bruderer's SA (BSA). A VLSI implementation of BSA is presented in [13].

It was observed in [10] that it suffices to calculate $4(N-1)$ new λ and Δ values at each iteration $\ell > 1$, instead of calculating all $N(N-1)$ values. More specifically, at iteration $(\ell+1)$, only $\lambda_{j,s^\ell}, \lambda_{j,t^\ell}, \lambda_{s^\ell,j}$, and $\lambda_{t^\ell,j}$ and the corresponding Δ values need to be calculated, where s^ℓ and t^ℓ denote the indices selected by Step 3 at iteration ℓ. This reduces the complexity of the λ and Δ steps to $\mathcal{O}(N)$. However, this idea still requires memory to store $N(N-1)$ λ and Δ values, and the maximum Δ step has to be performed over $N(N-1)$, leading to a per-iteration computational complexity of $\mathcal{O}(N^2)$. Moreover, to the best of our knowledge this idea has not been applied to a VLSI architecture to date.

III. Simplified Seysen's Algorithm

In this section, we described some additional simplifications to Seysen's algorithm, which lead to the efficient hardware architecture described in Section IV. Moreover, we evaluate the impact of these simplifications on the lattice reduction performance of the SA algorithm as well as on its computational complexity.

A. Further Restriction of λ Coefficients

The first simplification we propose is to further restrict the $\lambda_{i,j}$ coefficients to $\lambda_{i,j} \in \{\alpha + \beta j\},\ \alpha, \beta \in \{-1, 0, +1 : \alpha \neq 0 \veebar \beta \neq 0\}$ (i.e., α and β can not both be non-zero at the same time). In order to achieve this, we evaluate (22) separately for the real and the imaginary part of $\lambda_{i,j}$, and if $\Re(\lambda_{i,j}) \neq 0$ and $\Im(\lambda_{i,j}) \neq 0$, we set one of the two parts to zero. In our simulations, we observed that randomly setting one part to zero and always setting the real (or imaginary) part to zero lead to the same performance. In the VLSI architecture described in Section IV, we adopt the latter method as it is simpler to implement in hardware.

This restriction simplifies several conditional additions to conditional selections, which can be efficiently implemented in hardware by simple multiplexers. More specifically, (24) and (25) are further simplified to

$$\Re(x) = \begin{cases} \Re(\lambda)\Re(G), & \Re(\lambda) \neq 0, \Im(\lambda) = 0, \\ \Im(\lambda)\Im(G), & \Re(\lambda) = 0, \Im(\lambda) \neq 0, \\ 0, & \Re(\lambda) = 0, \Im(\lambda) = 0, \end{cases} \tag{26}$$

$$\Im(x) = \begin{cases} \Re(\lambda)\Im(G), & \Re(\lambda) \neq 0, \Im(\lambda) = 0, \\ -\Im(\lambda)\Re(G), & \Re(\lambda) = 0, \Im(\lambda) \neq 0, \\ 0, & \Re(\lambda) = 0, \Im(\lambda) = 0, \end{cases} \tag{27}$$

since the third case of (24) and (25) can never occur. This simplification saves two real additions per matrix element update (one for the real part and one for the imaginary part). Moreover, several computations in the λ and Δ calculation steps can be simplified similarly.

B. Approximate Maximum Selection

Recall that s^ℓ and t^ℓ denote the selected indices at Step 3 of iteration ℓ. Similarly, we denote the Δ values calculated at iteration ℓ by Δ^ℓ. Let $\mathcal{D}^{\ell+1}$ be the set

$$\mathcal{D}^{\ell+1} = \left\{ \max_{j \neq s^\ell} \Delta_{j,s^\ell}^{\ell+1}, \max_{j \neq t^\ell} \Delta_{j,t^\ell}^{\ell+1}, \max_{j \neq s^\ell} \Delta_{s^\ell,j}^{\ell+1}, \max_{j \neq t^\ell} \Delta_{t^\ell,j}^{\ell+1} \right\}. \tag{28}$$

Then, the maximum Δ value at iteration $\ell + 1$ can be equivalently calculated as

$$\Delta_{\max}^{\ell+1} = \max \left\{ \mathcal{D}^{\ell+1}, \max_{\substack{i,j \in \{1,\ldots,N\}, i \neq j, \\ i \neq s^\ell, i \neq t^\ell, j \neq s^\ell, j \neq t^\ell}} \Delta_{i,j}^\ell \right\}. \tag{29}$$

In words, the maximum Δ value at iteration $\ell + 1$ is either equal to the maximum of the $4(N-1)$ new Δ values, or to the maximum of the old Δ values when excluding rows s^ℓ and t^ℓ and columns s^ℓ and t^ℓ, which we call the *second* maximum. However, finding the second maximum requires first finding the first maximum of the $N(N-1)$ Δ values at each iteration, so this reformulation does not lead to a complexity reduction neither in the maximum finding step nor in the memory required to store the Δ values. Thus, we propose to approximate (29) as

$$\Delta_{\max}^{\ell+1} \approx \max \mathcal{D}^{\ell+1}, \tag{30}$$

by ignoring the second maximum. Since we are only ignoring a single candidate Δ value, the effect of this approximation on the performance of SSA is hopefully small.

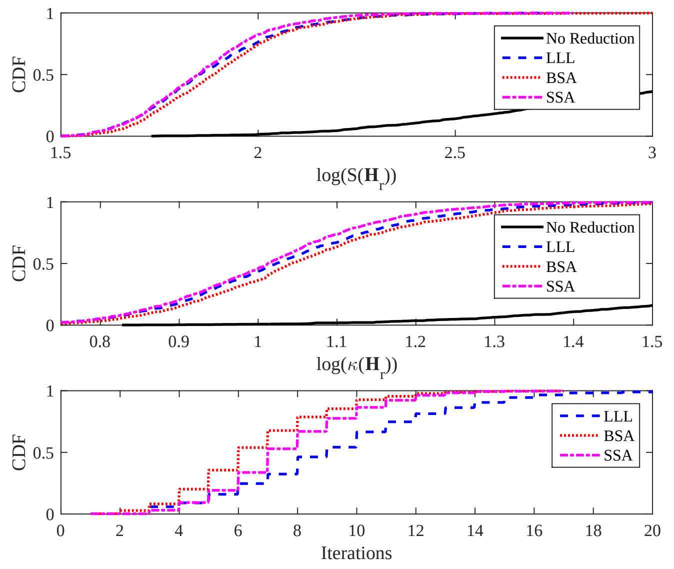

Fig. 1. Empirical cumulative density functions of $\log(S(\mathbf{H}_r))$, $\log(\kappa(\mathbf{H}_r))$, and the number of performed iterations for LLL, BSA ($K = 11$), and SSA for a 4×4 MIMO system.

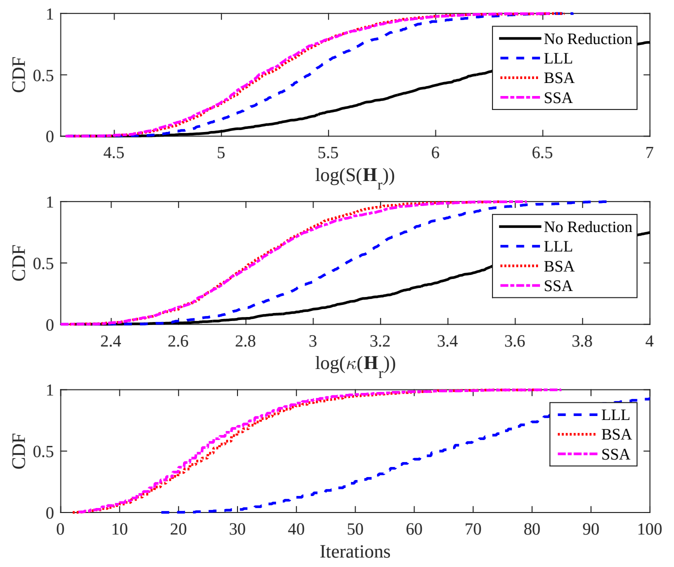

Fig. 2. Empirical cumulative density functions of $\log(S(\mathbf{H}_r))$, $\log(\kappa(\mathbf{H}_r))$, and the number of performed iterations for LLL, BSA ($K = 90$), and SSA for a 16×16 MIMO system.

With this approximation, the maximum finding step can be performed over $4(N-1)$ Δ values for any $\ell > 1$, meaning that the computational complexity of this step is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. Since all remaining steps of the algorithm also have computational complexity $\mathcal{O}(N)$, we conclude that the overall per-iteration computational complexity is $\mathcal{O}(N)$. We highlight that all steps for the first iteration still need to be performed over $\mathcal{O}(N^2)$ values, but this cost is amortized by the following iterations.

C. SSA Performance Evaluation

For a fair comparison between BSA and SSA, we selected K for BSA so that the lattice reduction performance of SSA is at least as good as that of BSA. Specifically, for a 4×4 system we selected $K = 11$ ($K_{\max} = 12$) and for a 16×16 system we have $K = 90$ ($K_{\max} = 240$). As can be seen in Fig. 1 and Fig. 2, the performance of BSA and SSA is indeed similar in terms of Seysen's metric $S(\mathbf{H}_r)$ and the condition number $\kappa(\mathbf{H}_r)$, but SSA requires slightly more iterations to converge. We also present the performance of LLL and we note that both BSA and SSA outperform LLL in all considered performance metrics for the 16×16 system, while only SSA outperforms LLL for the 4×4 system. Since it is difficult to directly relate Seysen's metric and the condition number with the performance of an LR-aided MMSE detector, in Fig. 3 we also present simulation results for a 4×4 and a 16×16 MIMO system using a 64-QAM constellation that demonstrate similar results.

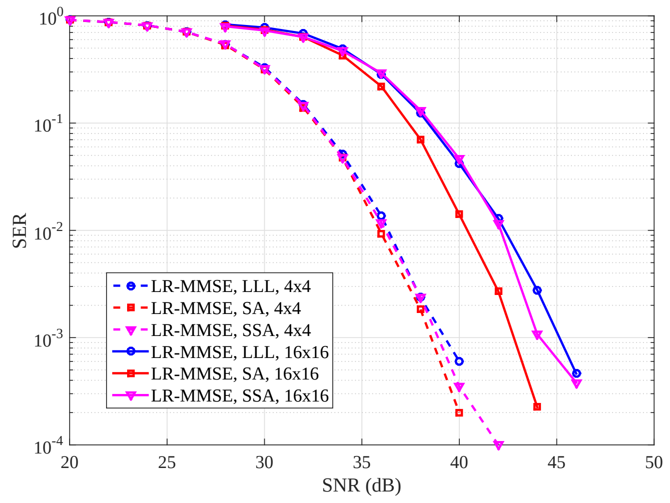

Fig. 3. Symbol error rate (SER) comparison of an LR-aided MMSE detector when using LLL, SA, and SSA for lattice reduction.

TABLE I. COMPLEXITY COMPARISON BETWEEN SA, BSA, AND SSA.

	4 × 4			16 × 16		
	SA	BSA	SSA	SA	BSA	SSA
Additions	216	152	84	3552	976	372
Multiplications	306	88	80	5058	720	464
Divisions	24	0	0	480	0	0
Δ values	12	11	10	240	90	58

D. SSA Complexity Comparison

In Table I, we compare the per-iteration complexity of SA, BSA, and SSA in terms of the required number of real additions and real multiplications, as well as the number of Δ values that need to be compared in order to find the maximum. In order to evaluate the complexity of BSA, we use the K values mentioned in the previous section for a fair comparison. We observe that SSA requires 45% and 62% fewer additions than BSA for $N = M = 4$ and $N = M = 16$, respectively. Moreover, both the multiplications and the number of candidate Δ values are reduced by 9% and 36% with respect to BSA for $N = M = 4$ and $N = M = 16$, respectively. When comparing SSA with the original SA, we see that SSA requires 61% and 90% fewer additions, and 74% and 91% fewer multiplications than SA for $N = M = 4$ and $N = M = 16$, respectively.

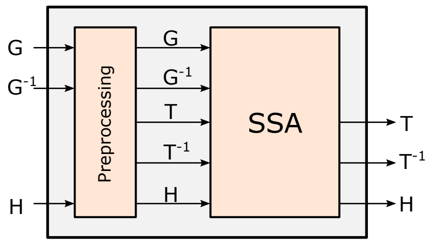

Fig. 4. High-level view of the proposed top-level architecture. The handshake signals for input and output are omitted for simplicity.

IV. VLSI Architecture and Implementation Results

A. VLSI Architecture

In this section, we present a VLSI architecture that implements the proposed SSA algorithm. Our architecture consists of a pre-processing block, which implements the first SSA iteration with complexity $\mathcal{O}(N^2)$, and an SSA block, which performs the remaining operations with complexity $\mathcal{O}(N)$. As in [13], we do not consider the matrix-by-matrix multiplication block for the computation of $\mathbf{G}$ and the matrix inversion block for the computation of $\mathbf{G}^{-1}$ as part of the LR core, as they are in any case required to implement an MMSE detector. The two blocks have dedicated memories and operate in a pipelined fashion, so that a new channel matrix can be processed in the pre-processing block while the previous channel matrix is being processed in the SSA block. An overview of the proposed top-level architecture is presented in Fig. 4.

1) Pre-processing block: The pre-processing block consist of two pipeline stages, each requiring one clock cycle. The first stage calculates $(N-1)$ $\lambda_{i,j}$ values in parallel an the second stage calculates $(N-1)$ $\Delta_{i,j}$ values in parallel and updates the maximum Δ value. When all values have been processed, the $\mathbf{G}$, $\mathbf{G}^{-1}$, $\mathbf{T}$, and $\mathbf{T}^{-1}$ matrices are updated in a single clock cycle. Thus, the overall latency of the pre-processing block, measured in clock cycles, is

$$L_{\text{PP}} = 2N + 1. \tag{31}$$

2) SSA Block: The SSA block operates on the updated $\mathbf{G}$, $\mathbf{G}^{-1}$, $\mathbf{T}$, and $\mathbf{T}^{-1}$ matrices coming from the pre-processing block. It consists of four pipeline stages, namely the λ stage, the Δ stage, the maximum stage, and the update stage, as well as memories to store all required matrices. Each pipeline stage requires one clock cycle. An overview of the SSA block architecture is presented in Fig. 5.

a) λ Stage: The λ stage contains four λ blocks, each containing $(N-1)$ λ units, in order to calculate all $\lambda_{j,s}, \lambda_{j,t}, \lambda_{s,j}$, and $\lambda_{t,j}$ values in parallel. We note that with this approach $\lambda_{s,t}$ and $\lambda_{t,s}$, are calculated twice, but it leads to a much more regular architecture.

b) Δ Stage: Similarly, the Δ stage contains four Δ blocks, each containing $(N-1)$ Δ units, in order to calculate all $\Delta_{j,s}, \Delta_{j,t}, \Delta_{s,j}$, and $\Delta_{t,j}$ values in parallel using the λ values calculated by the lambda stage.

c) Maximum Stage: The maximum stage is responsible for finding the maximum of the Δ values produced by the Δ stage, as well as output the corresponding $\{s,t\}$ indices.

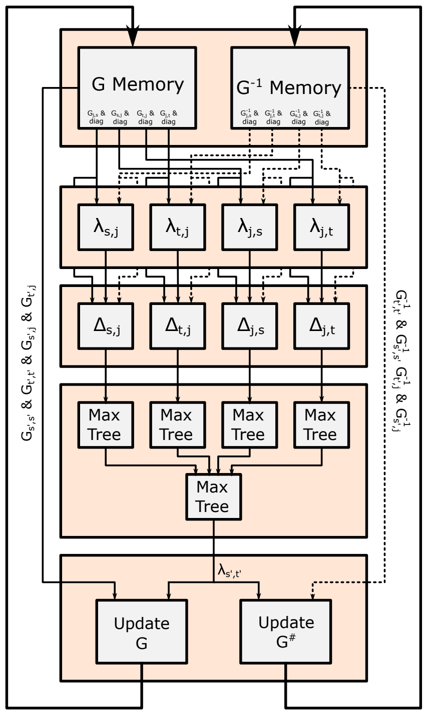

Fig. 5. High-level view of the proposed SSA block architecture. The $\mathbf{T}$ and $\mathbf{T}^{-1}$ memories and update units are omitted for simplicity.

d) Update Stage: The matrix update stage updates the row and column of $\mathbf{G}$, $\mathbf{G}^{-1}$, and $\mathbf{T}$, $\mathbf{T}^{-1}$ (and optionally, the $\mathbf{H}$ matrix), that was selected by the maximum stage.

When processing a single channel matrix at a time, each pipeline stage is idle for 75% of the duration of each iteration. In order to increase the hardware utilization of the SSA block, we instantiate four dedicated $\mathbf{G}$, $\mathbf{G}^{-1}$, $\mathbf{T}$, and $\mathbf{T}^{-1}$ memories (and optionally, four $\mathbf{H}$ memories) so that we can process four channel matrices in parallel. Due to the presence of a significant number of multipliers, our design is logic-dominated. Thus, processing four matrices in parallel increases the area only slightly while increasing the throughput by four times, leading to an overall increase in the hardware efficiency.

Similarly to [11], the SSA block implements early-termination based on $\Delta_{i,j}$. Specifically, we stop processing a given channel matrix when all $\Delta_{i,j} = 0$, or, equivalently, when $\max \Delta_{i,j} = 0$, which is a value readily available from the maximum stage. However, to maintain regular pipeline timing we keep the channel matrix in memory for the maximum number of iterations, denoted by $\ell_{\max}$. Thus, the early-termination function is mainly intended to reduce the power consumption and not to increase the throughput. The latency of the SSA block, measured in clock cycles, is

$$L_{\text{SSA}} = 4\left(\ell_{\max} - 1\right). \tag{32}$$

B. Quantization Parameters

Our architecture has several distinct quantization bit-widths for the quantities involved in the computations. More specifically for the elements of $\mathbf{G}$ and $\mathbf{G}^{-1}$, we use Q_G quantization bits. For the elements of $\mathbf{T}$ and $\mathbf{T}^{-1}$ we use Q_T quantization bits. Our architecture implements the reduced-width λ and

TABLE II. SYNTHESIS RESULTS

	4×4	8×8	16×16
Area (kGE)	108	318	981
Frequency (MHz)	1000	1000	1000
Iterations	12	24	36
Throughput (MMatrices/s)	91	45	23
Efficiency (MMatrices/s/kGE)	0.843	0.142	0.023

Δ calculation also employed in [13]. More specifically, the computations of λ and Δ are performed using only $Q_{\lambda,\Delta}$ quantization bits of the Q_G bits that are used to represent the elements of $\mathbf{G}$ and $\mathbf{G}^{-1}$. Finally, in case the $\mathbf{H}$ memory and update units are instantiated, the elements of $\mathbf{H}$ are stored using Q_H quantization bits.

C. Throughput

The SSA block can provide one output every $(\ell_{\max} - 1)$ clock cycles. If the pre-processing block can provide the SSA block with input at the same (or higher) rate, then the throughput is dictated by the SSA block. Otherwise, it is dictated by the pre-processing block. More specifically, the throughput, measured in MMatrices/s, can be calculated as

$$T = \frac{f_{\text{clk}}}{\max\{\ell_{\max} - 1, 2N + 1\}}, \tag{33}$$

where f_{clk} denotes the clock frequency of the decoder measured in MHz.

D. Synthesis Results

We synthesized the proposed VLSI architecture using a 22 nm technology for a target operating frequency of 1 GHz. We used the same quantization parameters as [13] for fair comparison. More specifically, we used $Q_G = 13$, $Q_{\lambda,\Delta} = 8$, and $Q_T = 4$.

Most current implementations of LR cores (e.g., [4], [5], [6], [7], [12], [13]) focus on 4×4 MIMO systems. However, as future systems are likely to employ MIMO on a much more massive scale, we believe that it is important to examine how proposed architectures scale with the number of employed antennas. To this end, in Table II, we present synthesis results for a 4×4, an 8×8, and a 16×16 MIMO system. We observe that the area increases at a rate that is slightly higher than linear in the system size, mainly due to the $\mathbf{G}$, $\mathbf{G}^{-1}$, $\mathbf{T}$, and $\mathbf{T}^{-1}$ memories, whose area increases quadratically with the number of antennas. Moreover, larger systems require more iterations for the SSA algorithm to reach a good solution. Thus, as the system size grows, the throughput is reduced even though the number of cycles per iteration is constant. We observe that the reduced throughput combined with the increased area, lead to a significant reduction in hardware efficiency.

In Table III, we compare our architecture for a 4×4 MIMO system with the existing SA-based 4×4 MIMO LR implementations of [12] and [13] for the sake of completeness. As the results are based on vastly different technologies, we do not make an attempt to scale the results as any comparison will be speculative at best.

TABLE III. COMPARISON WITH EXISTING 4×4 MIMO SA IMPLEMENTATIONS

	This work	[12]	[13]
Technology	22 nm	65 nm	90 nm
Area (kGE)	108	67	112
Frequency (MHz)	1000	400	360
Iterations	12	n/a	12
Throughput (MMatrices/s)	91	0.44	15
Efficiency (MMatrices/s/kGE)	0.843	0.007	0.134

V. CONCLUSION

In this work, we presented an approximation to SA, called SSA, which was shown to reduce the per-iteration computational complexity from quadratic to linear. More specifically, SSA requires 45% and 62% fewer additions as well as 9% and 36% fewer multiplications than the existing BSA for $N = M = 4$ and $N = M = 16$, respectively, while achieving the same lattice reduction performance. Moreover, we described an efficient VLSI architecture for the implementation of SSA and we presented synthesis results for various system sizes in order to explore the scaling behavior of our architecture.

REFERENCES

[1] M. Taherzadeh, A. Mobasher, and A. Khandani, "LLL reduction achieves the receive diversity in MIMO decoding," *IEEE Trans. Inf. Theory*, vol. 53, no. 12, pp. 4801–4805, Dec. 2007.

[2] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982.

[3] Y. H. Gan, C. Ling, and W. H. Mow, "Complex lattice reduction algorithm for low-complexity MIMO detection," *arXiv:cs/0607078*, Jul. 2006, arXiv: cs/0607078. [Online]. Available: http://arxiv.org/abs/cs/0607078

[4] B. Gestner, W. Zhang, X. Ma, and D. Anderson, "VLSI implementation of a lattice reduction algorithm for low-complexity equalization," in *IEEE Int. Conf. Circ. and Systems for Commun. (ICCSC)*, May 2008, pp. 643–647.

[5] L. Bruderer, C. Studer, M. Wenk, D. Seethaler, and A. Burg, "VLSI implementation of a low-complexity LLL lattice reduction algorithm for MIMO detection," in *IEEE Int. Symp. Circ. and Systems (ISCAS)*, May 2010, pp. 3745–3748.

[6] C.-F. Liao and Y.-H. Huang, "Power-saving 4×4 lattice-reduction processor for MIMO detection with redundancy checking," *IEEE Trans. Circ. and Systems II: Express Briefs*, vol. 58, no. 2, pp. 95–99, Feb. 2011.

[7] M. Shabany, A. Youssef, and G. Gulak, "High-Throughput 0.13- CMOS Lattice Reduction Core Supporting 880 Mb/s Detection," *IEEE Trans. Very Large Scale Int. (VLSI) Systems*, vol. 21, no. 5, pp. 848–861, May 2013.

[8] M. Seysen, "Simultaneous reduction of a lattice basis and its reciprocal basis," *Combinatorica*, vol. 13, no. 3, pp. 363–376, 1993.

[9] W. Zhang, X. Ma, and A. Swami, "Designing low-complexity detectors based on Seysen's algorithm," *IEEE Trans. Wireless Commun.*, vol. 9, no. 10, pp. 3301–3311, Oct. 2010.

[10] D. Seethaler, G. Matz, and F. Hlawatsch, "Low-complexity MIMO data detection using Seysen's lattice reduction algorithm," in *IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, vol. 3, Apr. 2007, pp. III–53–III–56.

[11] L. Bruderer, C. Senning, and A. Burg, "Low-complexity Seysen's algorithm based lattice reduction-aided MIMO detection for hardware implementations," in *Asilomar Conf. on Signals, Systems and Computers*, Nov. 2010, pp. 1468–1472.

[12] D. Wu, J. Eilert, and D. Liu, "A programmable lattice-reduction aided detector for MIMO-OFDMA," in *IEEE Int. Conf. Circ. and Systems for Commun. (ICCSC)*, May 2008, pp. 293–297.

[13] C. Senning, L. Bruderer, J. Hunziker, and A. Burg, "A lattice reduction-aided MIMO channel equalizer in 90 nm CMOS achieving 720 Mb/s," *IEEE Trans. Circ. and Systems I: Regular Papers*, vol. 61, no. 6, pp. 1860–1871, Jun. 2014.