Implementation of a High-throughput and Area-efficient MIMO Detector Based on Modified Dijkstra's Search

Tae-Hwan Kim and In-Cheol Park
Department of EE, KAIST
Yuseung-gu, Daejeon, Republic of Korea

Abstract— **This paper presents a VLSI implementation of a high-throughput and area-efficient MIMO detector. We propose a modified Dijkstra's algorithm and a pre-calculation technique to improve the throughput by allowing overlapped processing. In addition, we propose a simple approximation of L^2-norm to reduce the computational complexity without degrading the error performance noticeably. A MIMO detector based on the proposed algorithm is implemented using a 0.18-μm CMOS technology, which occupies 0.49 mm^2 with 25.1K equivalent gates and shows a throughput of over 300 Mbps.**

Index Terms— **Multi-input multi-output, maximum-likelihood (ML) detection, sphere decoding, very large scale integration (VLSI), wireless communications.**

I. Introduction

The MIMO communication system is associated with multiple spatial streams to extend channel capacity [1]. The large channel capacity can be used to boost the data rate, but much complicated signal processing is inevitably required because of the multiplicity of spatial streams as well as the interferences among them.

Aiming at a high-throughput, area-efficient VLSI architecture, this paper proposes new methods to detect the MIMO symbols, and presents an implementation based on them. The proposed MIMO detector is the first realization of the sphere decoder (SD) based on Dijkstra's algorithm. To enhance the throughput without incurring a significant degradation of the error performance, the original Dijkstra's algorithm is modified to enable overlapped processing. To improve the throughput further, we pre-calculate the best candidate to be expanded in the next iteration. Due to the algorithmic modification as well as the pre-calculation, the critical path delay of the proposed architecture is reduced to the delay of tree expansion. In addition, we propose a simple approximation of L^2-norm to reduce both the hardware complexity and the critical path delay. The proposed MIMO detector is implemented in a 0.18-μm CMOS technology, which occupies 0.49 mm^2 and shows a throughput of over 300 Mbps in the environment of high signal-to-noise ratio (SNR).

The rest of the paper is organized as follows. Section II describes the system model and the original Dijkstra's algorithm for the MIMO detection. Section III proposes the modified Dijkstra's algorithm and the pre-calculation technique. Section IV presents the architecture of the proposed MIMO detector with a simple approximation of L^2-norm. In Section V, the performance of the proposed algorithm is evaluated, and the implementation results of the proposed MIMO detector is compared with the previous works. Finally, concluding remarks are made in Section VI.

II. MIMO Detection

A. System Model

Let us consider a $N_T \times N_R$ MIMO communication system that employs N_T transmitting antennas and N_R receiving antennas. The corresponding baseband-equivalent system can be modeled as

$$\mathbf{y}_c = \mathbf{H}_c \cdot \mathbf{x}_c + \mathbf{n}_c, \tag{1}$$

where $\mathbf{y}_c$ is the $N_R \times 1$ received symbol vector, $\mathbf{H}_c$ is the $N_R \times N_T$ channel matrix, $\mathbf{x}_c$ is the $N_T \times 1$ transmitted symbol vector, and $\mathbf{n}_c$ is the $N_R \times 1$ additive noise vector. Each symbol in $\mathbf{x}_c$ is drawn from a constellation. (1) can be decomposed into the real domain as

$$\mathbf{y} = \mathbf{H} \cdot \mathbf{x} + \mathbf{n}. \tag{2}$$

In (2), $\mathbf{H}$, $\mathbf{y}$, $\mathbf{x}$, and $\mathbf{n}$ are defined as

$$\mathbf{H} = \begin{bmatrix} \mathrm{Re}(\mathbf{H}_c) & -\mathrm{Im}(\mathbf{H}_c) \\ \mathrm{Im}(\mathbf{H}_c) & \mathrm{Re}(\mathbf{H}_c) \end{bmatrix}, \tag{3}$$

$$\mathbf{y} = [\mathrm{Re}(\mathbf{y}_c^T)\ \mathrm{Im}(\mathbf{y}_c^T)]^T, \tag{4}$$

$$\mathbf{x} = [\mathrm{Re}(\mathbf{x}_c^T)\ \mathrm{Im}(\mathbf{x}_c^T)]^T, \tag{5}$$

$$\mathbf{n} = [\mathrm{Re}(\mathbf{n}_c^T)\ \mathrm{Im}(\mathbf{n}_c^T)]^T, \tag{6}$$

where Re($\mathbf{v}$) and Im($\mathbf{v}$) denote the real and the imaginary part of $\mathbf{v}$, respectively. In this paper, we will take into account the real-valued model expressed in (2), assuming that $N = 2N_T = 2N_R$ without loss of generality.

B. MIMO Detection Using Dijkstra's Algorithm

Given $\mathbf{y}$ and $\mathbf{H}$, the maximum-likelihood (ML) MIMO detection is to find the optimal $\mathbf{x}$ such that

$$\arg\min_{\mathbf{x} \in \mathbf{O}^N} \left\| \mathbf{y} - \mathbf{H} \cdot \mathbf{x} \right\|^2, \tag{7}$$

where $\mathbf{O}$ is the constellation in the real-valued model. In 16-QAM, for example, $\mathbf{O}$ = {-3, -1, +1, +3}. By applying the QR-decomposition to $\mathbf{H}$, (7) can be simplified to

$$\arg\min_{\mathbf{x} \in \mathbf{O}^N} \left\| \mathbf{y} - \mathbf{Q} \cdot \mathbf{R} \cdot \mathbf{x} \right\|^2 = \arg\min_{\mathbf{x} \in \mathbf{O}^N} \left\| \hat{\mathbf{y}} - \mathbf{R} \cdot \mathbf{x} \right\|^2, \tag{8}$$

where $\mathbf{Q}$ is an orthogonal matrix, $\mathbf{R}$ is an upper triangular matrix, and $\hat{\mathbf{y}}$ is $\mathbf{Q}^T\mathbf{y}$. In the optimal solution in (8), the cost function is proportional to the square of Euclidean distance, $\|\hat{\mathbf{y}}\text{-}\mathbf{R}\mathbf{x}\|^2$, which can be recursively calculated as

$$b_n = \hat{y}_n - \sum_{i=n+1}^{N} r_{ni} x_i \,, \tag{9}$$

$$PED_n = PED_{n+1} + \left| b_n - r_{nn} x_n \right|^2 , \tag{10}$$

where $\hat{y}_n$ is the n-th element of $\hat{\mathbf{y}}$, r_{ij} is the (i, j)-th element of $\mathbf{R}$, x_i is the i-th element of $\mathbf{x}$, and PED_n is the partial Euclidean distance (PED) calculated at the n-th layer. Starting from PED_{N+1}=0, (10) is recursively calculated until it reaches the final cost, PED_1. Note that PED_n is always not less than PED_{n+1} because the second term on the right side in (10) is non-negative.

After constructing a decoding tree where every node except the root corresponds to a symbol in the symbol vector, the symbol detection can be regarded as a tree search problem. Dijkstra's algorithm to find the shortest path in a graph can be applied to the MIMO detection [2][3]. Fig. 1(a) shows the conceptual flow of the algorithm, where a candidate associated with the best metric is selected in a greedy manner and it is expanded into the lower layer. When the algorithm is applied to the MIMO detection, each candidate corresponds to a node in the decoding tree, and its metric is the PED. The detailed algorithm is described below, and an example of the MIMO detection based on this algorithm is shown in Fig. 1(b).

1) **C** is the candidate set, and its size, |**C**|, is constrained to s. Initially, **C** contains only the root node of the decoding tree and its PED is set to 0.
2) *CurrentBest*←MinPED(**C**), where MinPED(**C**) selects the candidate having the smallest PED in **C**. If *CurrentBest* is in the lowest layer, stop the algorithm.
3) ***Children***←TreeExpand(*CurrentBest*), where TreeExpand(*CurrentBest*) is the tree-expansion from *CurrentBest* according to (9) and (10). The size of Children is |**O**|.
4) **C**←Evict((**C**-{*CurrentBest*}) ⋃ **Children**, s) where Evict(**A**, s) removes bad candidates from **A** if |**A**|>s. Go to 2).

To generate the optimal solution, the original Dijkstra's algorithm has no constraint of |**C**|, which means s can be infinite. However, s should be constrained in order to make the hardware implementation feasible, and the performance is reported to be near-optimal if s is moderately large [2][3]. In contrast to the general SD [4], this algorithm does not need any enumeration to determine the visiting order of sub-trees. In terms of the number of visited nodes, this algorithm shows much lower complexity [2] than depth-first search (DFS) SD or K-best [5], meaning that it is more suitable for achieving the high-throughput. However, there are little studies on the efficient realization of this algorithm.

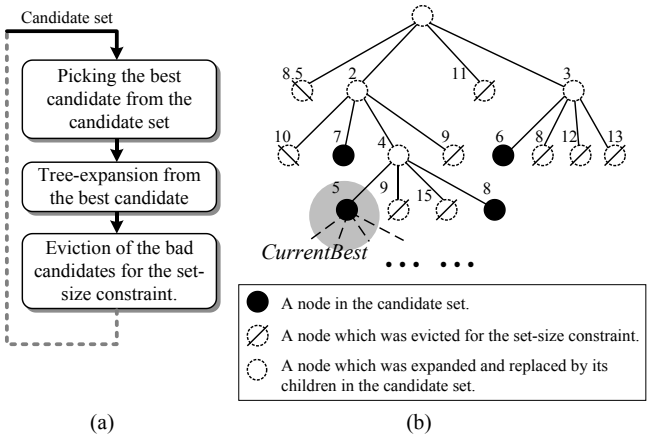


Fig. 1. (a) Conceptual flow of the original Dijkstra's algorithm for MIMO symbol detection. (b) A detection example where the candidate set size is constrained to 4 and the number of a node represents the PED of the node.

III. Proposed Algorithm

A. Modified Dijkstra's Algorithm for Overlapped Processing

The original algorithm introduced in Section II is iterative, and each of the iteration consists of the three basic steps as shown in Fig. 1(a). In the first and third steps, we need to sort the candidate set or to do minimum/maximum operations. Additionally, as the eviction of the bad candidate is performed for the union of the previous candidate set and the new candidates generated by the tree-expansion, we cannot perform the tree-expansion in parallel with the eviction. To achieve a high-throughput, therefore, we need to develop a deep pipeline architecture that necessitates a large number of pipeline registers to store the intermediate results and the candidates. In addition, the deep pipeline lengthens the detection latency.

Observing the eviction process of the bad candidates in the original algorithm, we can see that there is only a little difference between two lists of surviving candidates obtained by performing the eviction process before and after the tree-expansion. In other words, the surviving candidates in Evict((**C**-{*CurrentBest*}) ⋃ **Children**, s) are similar to those in Evict((**C**-{*CurrentBest*}), s-|**O**|). Let us suppose a candidate that is not evicted for the former eviction process but evicted for the latter eviction process. It is very unlikely for the candidate to survive as a final solution. This algorithmic behavior becomes clear if s is large enough to guarantee the near-optimal performance.

Motivated by this observation, the last two steps in Fig. 1(a) are performed in a completely overlapped manner in the proposed algorithm. In other words, the eviction is processed in parallel with the tree expansion. In the proposed algorithm, the candidates are classified into two groups each of which is kept in a separate list. One list called the expanded list has only the candidates generated by the tree-expansion in the previous iteration, and the other list called the candidate list contains the surviving candidates. Fig. 2 shows the conceptual flow of the proposed algorithm. The current best candidate is first selected

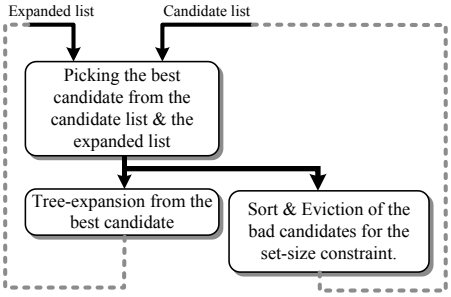


Fig. 2. Conceptual flow of the proposed algorithm.

among the candidates in the two lists. The candidate list is updated by merging the expanded list and performing eviction. Note that the expanded list was generated by the tree-expansion of the previous iteration. Hence, the tree-expansion from the current best candidate can be performed in parallel with the sorting and eviction.

To reduce the sorting complexity, the strictness of sorting can be relaxed by employing the distributed sorting technique proposed in [6]. The following is the modified Dijkstra's algorithm proposed for the MIMO detection, where the candidate list is again divided into two sub-lists, **A** and **B**:

1) **A** and **B** are two candidate lists whose sizes are constrained to $(s\text{-}|\mathbf{O}|)/2$. **E** is the expanded list having the candidates generated by the tree-expansion in the very previous iteration. The candidates in these lists are sorted according to the PED, i.e., $A_i.PED \le A_j.PED$, $B_i.PED \le B_j.PED$, and $E_i.PED \le E_j.PED$ for $1 \le i \le j$, where A_i is the i-th candidate in **A**, and $A_i.PED$ is the PED of A_i. Initially, **A**=**B**=Ø, and **E** contains only the root node whose PED is set to 0.
2) *CurrentBest*←MinPED($\{A_1,B_1,E_1\}$). In this computation, A_1 is not considered if **A** = Ø. Similarly, B_1 is not considered if **B** = Ø. If *CurrentBest* is in the lowest layer, stop the algorithm.
3) **A**←SortEvict(**A**$\bigcup\{E_1,E_3,\ldots\}$ - {*CurrentBest*}, $(s - |\mathbf{O}|)/2$). **B**←SortEvict(**B**$\bigcup\{E_2,E_4,\ldots\}$ - {*CurrentBest*}, $(s - |\mathbf{O}|)/2$). SortEvict(**X**, s) is to sort **X** by the PED and to evict the bad candidates if $|\mathbf{X}|>s$.
4) **E**←TreeExpandSort(*CurrentBest*), where TreeExpandSort(*CurrentBest*) is the same as TreeExpand(*CurrentBest*) except that the resulting **E**=$\{E_1,\ldots E_{|\mathbf{O}|}\}$ is sorted by the PED. Go to 2).

TreeExpandSort(·) in the above algorithm produces the sorted list of the expanded candidates. In the real-valued MIMO system, the sorting can be simply performed by comparing b_n with r_{nn} [7]. This can be performed before the addition of the PED increment in (10), so the delay additionally needed to sort the expanded candidates is negligible.

In 3) of the above algorithm, the candidates in **E** are distributed into two candidate lists for SortEvict operations. Note that **E** is divided into $\{E_1, E_3, ...\}$ and $\{E_2, E_4, ...\}$ rather than $\{E_1, ..., E_{|\mathbf{O}|/2}\}$ and $\{E_{|\mathbf{O}|/2+1}, ..., E_{|\mathbf{O}|}\}$. This arrangement prevents good candidates from being clustered in one of the two lists, and is effective in relieving the performance degradation caused by the distributed sorting.

SortEvict operation that merges two sorted lists into another sorted one requires less computation than the usual sorting. In the proposed algorithm, its complexity is further lowered by reducing the number of elements to be sorted. Two SortEvict operations each of which merges two sorted lists whose sizes are $(s\text{-}|\mathbf{O}|)/2$ and $|\mathbf{O}|/2$ are required in the proposed algorithm, whereas the original algorithm requires one sorting operation to merge two sets whose sizes are s and $|\mathbf{O}|$. Though the distributed sorting can be employed in the original algorithm, two sorting operations are required to merge two sets whose sizes are $s/2$ and $|\mathbf{O}|/2$.

The overlapped processing of eviction and tree expansion allows us to increase the operating frequency without employing a deep-pipeline architecture. As the overlapped processing is enabled by modifying the original algorithm, it does not induce any additional computations. However, the error performance may be degraded a little, because the proposed algorithm modifies the original algorithm to overlap the processing. Experimental results show that the performance degradation is negligible if s is as large as 16, as will be presented in Section V.

B. Pre-decision of the Best Candidate by Difference Comparisons

By using the algorithm proposed in the previous subsection, each iteration can be composed with two steps depicted in Fig. 2. The first step picks the best candidate, which was formally expressed as *CurrentBest*←MinPED($\{A_1,B_1,E_1\}$), where A_1, B_1, and E_1 are the best candidates in the sorted lists of **A**, **B**, and **E**, respectively. This step can be achieved by simple comparisons, but cannot be parallelized with the following step, because the next step is dependent on the choice of the current best candidate, *CurrentBest*.

We can pre-calculate the comparisons to decide the next best candidate, *NextBest*, which will be expanded in the next iteration. Considering the current candidates and the expanded candidates generated by the tree-expansion, we can pre-decide *NextBest* as follows:

$$NextBest \leftarrow \begin{cases} \text{MinPED}(\{A_2, B_1, E_1, P_1\}) & \text{if} \quad CurrentBest = A_1 \\ \text{MinPED}(\{A_1, B_2, E_1, P_1\}) & \text{if} \quad CurrentBest = B_1 \\ \text{MinPED}(\{A_1, B_1, E_2, P_1\}) & \text{if} \quad CurrentBest = E_1 \end{cases}, \quad (11)$$

where P_1 is the best child expanded from *CurrentBest*. Let $\text{MinPED}_{\text{diff}}(\mathbf{X},C)$ represent the candidate in **X** that has the smallest PED difference from C. In other words, if Y=$\text{MinPED}_{\text{diff}}(\mathbf{X},C)$, Y is closest to C in terms of PED among all the candidates in **X**. If $C.PED$ is not greater than the PED of every candidate in **X**, $\text{MinPED}_{\text{diff}}(\mathbf{X},C)$ is equivalent to MinPED(**X**). As *CurrentBest* is the best candidate in the current candidate lists, it is clear that the PED of *CurrentBest* is not greater than those of A_1, A_2, B_1, B_2, E_1, and E_2. As the new candidates are generated by expanding *CurrentBest* and the PED increases monotonically in (10), the PED of *CurrentBest* is not greater than that of P_1, either. Therefore, the condition holds if C is *CurrentBest*, which means that we can replace MinPED($\{V, W, Y, Z\}$) in (11) with $\text{MinPED}_{\text{diff}}(\{V, W, Y, Z\}$, *CurrentBest*), where $\{V, W, Y, Z\}$ corresponds to one of $\{A_2, B_1, E_1, P_1\}$, $\{A_1, B_2, E_1, P_1\}$, and $\{A_1, B_1, E_2, P_1\}$ according to the case. Additionally, $\text{MinPED}_{\text{diff}}(\{V, W, Y, Z\}$, *CurrentBest*) can be calculated as

$$\text{MinPED}_{\text{diff}}(\{\text{MinPED}_{\text{diff}}(\ \{V,W,Y\},\ CurrentBest\ ),\ Z\},\ CurrentBest). \quad (12)$$

As Z in (12) is generated by expanding *CurrentBest*, the PED difference between *CurrentBest* and Z is equal to the PED increment corresponding to the second term in the right hand side of (10). Note that the PED increment is available before finishing the tree-expansion, and $\text{MinPED}_{\text{diff}}(\{V,W,Y\}$, *CurrentBest*) can be calculated in parallel with the calculation

of PED increment. Therefore, the pre-decision of *NextBest* can be performed in parallel with the tree-expansion without incurring any additional delay.

Fig. 3 shows the proposed algorithm associated with the pre-decision of the next best candidate. In this algorithm, picking the best candidate is just selecting the candidate indicated by the pre-decision calculated in the previous iteration. As a result, we can effectively eliminate the delay of the comparison needed to select *CurrentBest* at the cost of a little additional computation needed to compare the PED difference instead of the PED itself.

IV. Proposed MIMO Detector

This section presents the architecture of a MIMO detector developed based on the algorithms proposed in the previous section. The architecture aims at 4 × 4, 16-QAM MIMO systems. As we are taking into account the system expressed in (2), $N = 8$ in this case. To further reduce the computational complexity without sacrificing the performance, we propose a simple approximation of L^2-norm in the tree-expansion.

A. Overall Architecture

The overall architecture of the proposed MIMO detector is depicted in Fig. 4. The major component is the detection core that performs the iterative algorithm proposed in the previous section. In each cycle, it produces new candidates. As shown in Fig. 4, the candidates are kept in two sorted lists of candidates, $\mathbf{A}=\{A_1,A_2,\ldots A_6\}$, $\mathbf{B}=\{B_1,B_2,\ldots B_6\}$, and a sorted list of expanded candidates, $\mathbf{E}=\{E_1,E_2,E_3,E_4\}$. The best candidate to be expanded is selected by the *whichBest* signal calculated in the previous iteration. The tree-expansion unit expands the decoding tree from the current best candidate, and the permutation unit simply arranges the expanded candidates according to the permutation vector calculated by the ordering unit. The PED differences between the current best candidate and the other candidates which can be expanded in the next iteration are calculated in parallel with the tree-expansion, and the *whichBest* signal is calculated by comparing the minimum PED difference and the minimum PED increment of the expanded candidates.

Two merging units are required for the SortEvict operations, each of which produces a sorted list of size 6 by merging two sorted lists whose sizes are 6 and 2. Each merging unit is implemented by employing bitonic sorting network [8]. As

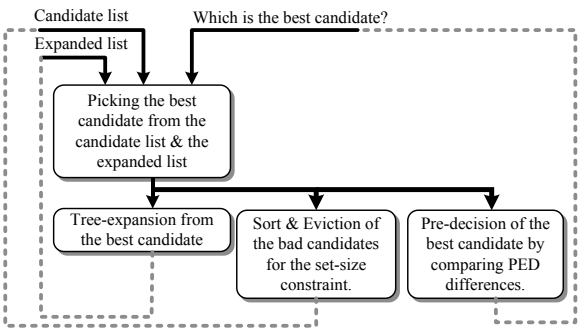


Fig. 3. Conceptual flow of the proposed algorithm based on the pre-calculation technique.

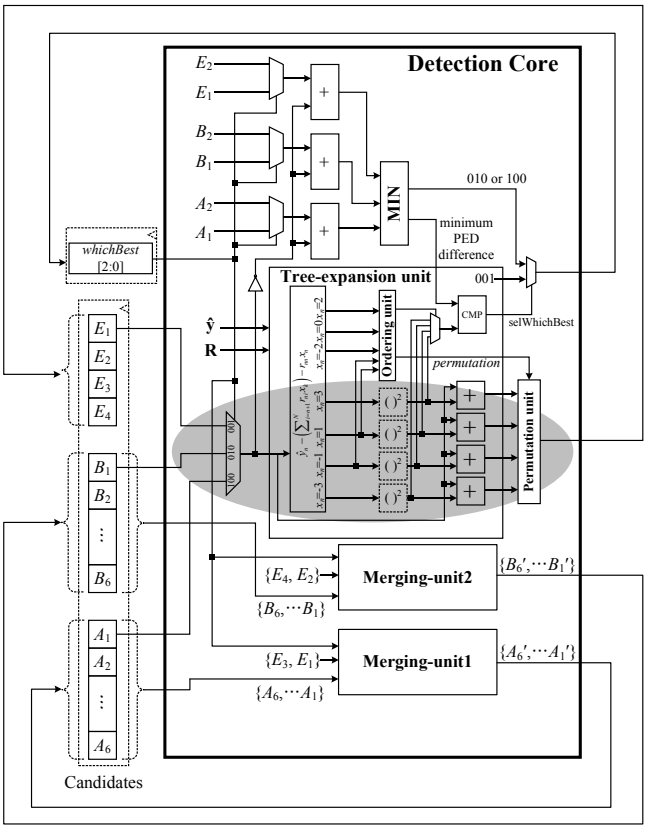


Fig. 4. Overall architecture of the proposed MIMO symbol detector with the overall critical path highlighted in gray.

highlighted in Fig. 4, the merging units are not included in the critical path, because they work in parallel to the tree expansion.

B. Tree-expansion with Simple Approximation of L^2-norm

In the tree-expansion unit, calculating the PED of a new candidate expanded from *CurrentBest* requires interference cancellation and norm computation as expressed in (9) and (10). The calculation of L^2-norm in ML detection requires high computational complexity because it has square operations. Some previous works have studied the use of other simple norms such as L^1-norm or L^∞-norm [7][9][10] to reduce the computational complexity at the cost of degrading the error performance.

In the MIMO detection, we deal with symbols corrupted by noise. To achieve an efficient implementation, in this case, it is more desirable to approximate the square operation as long as the monotonic property of the square operation is maintained. The square of an m-bit unsigned number p can be calculated as

$$(p[m\text{-}1]\cdot 2^{m-1} + p[m\text{-}2]\cdot 2^{m-2} + \ldots + p[1]\cdot 2^1 + p[0])^2. \quad (13)$$

This can be approximated as

$$p[m\text{-}1]\cdot 2^{2m-2} + p[m\text{-}2]\cdot 2^{2m-4} + \ldots + p[1]\cdot 2^2 + p[0], \quad (14)$$

which can be simply realized by inserting zeros between two adjacent bits in p as follows:

$$(p[m\text{-}1], 0, p[m\text{-}2], 0, \ldots, 0, p[1], 0, p[0]). \quad (15)$$

The zero insertion in (15) can be implemented by wiring without incurring any additional gate delay. Hence, the

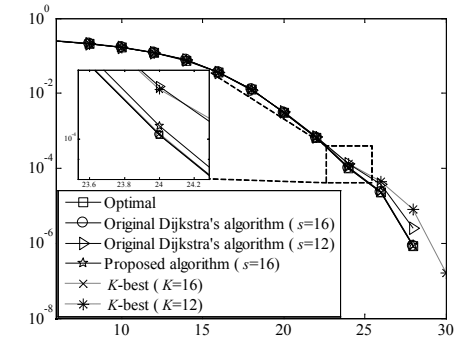
computational complexity to calculate the proposed approximation of L^2-norm is as low as those of L^1-norm and L^∞-norm. Additionally, it is obvious that the proposed approximate squaring is monotonic as the exact squaring is. Therefore, we can expect that it does not degrade the error performance noticeably.

V. EXPERIMENTAL RESULTS

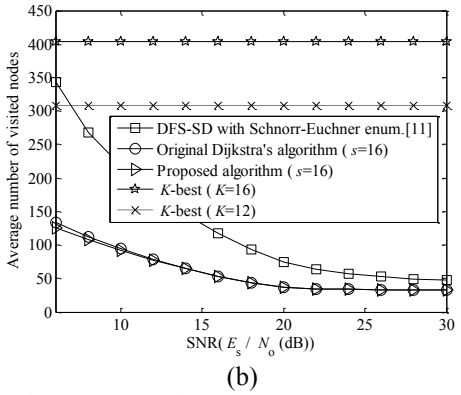
Fig. 5 shows the performances of the modified Dijkstra's algorithm proposed in Section III and those of others including the original Dijkstra's algorithm, *K*-best, and so on. As shown in Fig. 5(a), the BER performance degradation caused by the overlapped processing is very little even compared with the optimal result. With the same size of *s* that guarantees the near-optimal performance in the original algorithm, the performance degradation of the proposed algorithm is negligible as shown in Fig. 5(a). If the target BER is 10^{-4}, the performance gap between the proposed algorithm and the optimal one is only about 0.02dB in terms of SNR.

Fig. 5(b) shows the average number of visited nodes in the decoding tree. The less number of visits leads to the higher throughput. The average number of visits in the proposed algorithm is comparable to that of the original Dijkstra's algorithm, and both are much smaller than those of the DFS-SD and *K*-best.

Fig. 6 shows how the performances are affected by the

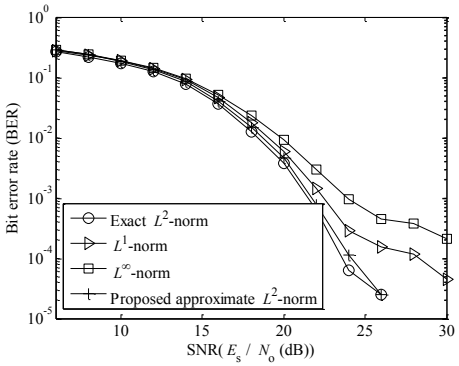


(a)

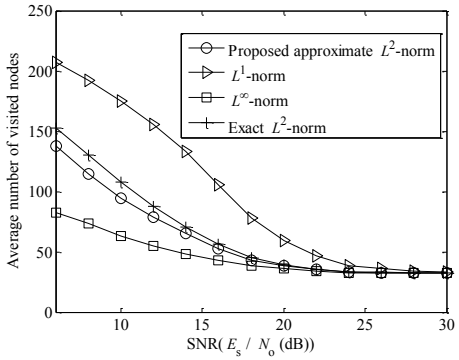


(b)

Fig. 5. Performance comparison for 4 x 4, 16-QAM MIMO symbol detection: (a) BER performance, and (b) the average number of visited nodes. The sorting in the original Dijkstra's algorithm is not relaxed in opposition to the proposed algorithm. The SNR per receiving antenna is defined as E_s/N_o, where E_s means the symbol energy and N_o means the noise variance.



(a)



(b)

Fig. 6. Performance of the proposed approximate L^2-norm: (a) BER performance, and (b) the average number of the visited nodes.

approximate L^2-norm proposed in Section IV. The performance of the proposed approximation is comparable to that of the exact L^2-norm for the whole range of the SNR, and much better than the other simple norms. Even though L^∞-norm reveals the smallest number of visits as shown in Fig. 6(b), the performance degradation resulting from L^∞-norm is somewhat severe as shown in Fig. 6(a).

The proposed MIMO detector is implemented in a 0.18-μm CMOS technology. The critical path delay of the proposed design is about 5.41 ns when estimated with reflecting the parasitic effects. Counting a 2-input NAND as one, the equivalent gate-count of the proposed MIMO detector is about 25.1K. The proposed MIMO detector occupies 0.49 mm^2.

The throughput of the proposed MIMO detector is estimated with taking into account the operating frequency and the average number of visited nodes. Since four 16-QAM symbols are transmitted per transaction in the target system and four nodes are visited by the tree-expansion in one cycle in the proposed architecture, the throughput is calculated as

$$\frac{16 \cdot (\text{operating frequency})}{(\text{average number of visited nodes})/4} \text{ bps}. \qquad (16)$$

The throughput of the proposed MIMO detector is shown in Fig. 7, where we can see that the throughput is about 300 Mbps for a SNR of 20 dB and about 360 Mbps for a SNR of 30 dB.

In Table I, the characteristics of the proposed MIMO detector are compared with those of previous works. In the proposed MIMO detector, the decoding tree is expanded in a

TABLE I. COMPARISONS WITH PREVIOUS WORKS

Architecture	[7]	[12]	[9]		Proposed
Configuration	4x4 16-QAM	4x4 16-QAM	4x4 16-QAM	4x4 16-QAM	4x4 16-QAM
Algorithm	K-Best (K=5)	K-Best SE [a] (K=5)	DFS-SD	DFS-SD	Modified Dijkstra's algorithm
Norm calculation	L^1-norm	L^2-norm	L^2-norm	L^∞-norm	Proposed approx. L^2-norm
Technology	0.25-μm CMOS	0.35-μm CMOS	0.25-μm CMOS	0.25-μm CMOS	0.18-μm CMOS
Gate count	68K	91K	117K	50K	25.1K
Area	N.A.	5.76 mm^2	N.A.	N.A.	0.49 mm^2
Frequency	132MHz	100MHz	51MHz	73MHz	181MHz
Throughput	424Mbps	53.3Mbps	73Mbps (@SNR=20dB)	169Mbps (@SNR=20dB)	302Mbps (@SNR=20dB)

[a] K-Best SE algorithm is a K-best algorithm modified by using the Schnorr-Euchner searching strategy [12].

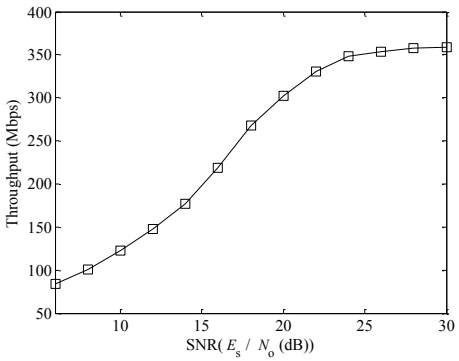


Fig. 7. Throughput and layout of the proposed MIMO symbol detector.

greedy manner, the sorting and eviction is distributed, and there is a single processing element. As a result, it can lower the hardware complexity significantly, and has the advantage of achieving high area efficiency as well as high throughput over the other detectors. In addition, the proposed architecture employs a simple approximation to reduce the computational complexity of L^2-norm and provide better performance than L^1-norm and L^∞-norm.

VI. CONCLUSION

Observing the behavior of the original Dijkstra's algorithm, we have proposed a new MIMO detection algorithm that can lead to high-throughput architecture and presented its efficient implementation. The original Dijkstra's algorithm is modified to allow computational steps in the critical path to be overlapped as much as possible. To improve the throughput further, the best candidate to be expanded next is pre-calculated. Additionally, we proposed a simple approximation of L^2-norm to reduce the computational complexity without severe performance degradation. A prototype MIMO detector based on the proposed architecture was implemented in a 0.18-μm CMOS technology. The chip integrating 25.1K equivalent gates occupies 0.49 mm^2 and its throughput is over 300 Mbps for high SNR.

REFERENCES

[1] D. Tse and P. Viswanath, "Fundamentals of wireless communication," *Cambridge University Press*, 2005.

[2] M. Myllyla, M. Juntti, and J. R. Cavallaro, "Implementation aspects of list sphere detector algorithms," *in Proc. of Global Telecommunications Conference*, pp.3915-3920, Nov. 2007.

[3] S. Bäro, J. Hagenauer, and M. Witzke, "Iterative detection of MIMO transmission using a list-sequential (LISS) detector," *in Proc. of International Conference on Communications*, vol.4, pp.2653-2657, May 2003.

[4] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. expected complexity," *IEEE Trans. on Signal Processing*, vol.53, no.8, pp.2806-2818, Aug. 2005.

[5] K.W. Wong, C.Y. Tsui, Cheng. Roger S.K. Cheng, and W. H. Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," *in Proc. of International Symposium on Circuits and Systems*, vol.3, pp.273-276, May 2002.

[6] S. Chen, T. Zhang, and Y. Xin, "Relaxed K-best MIMO signal detector design and VLSI implementation," *IEEE Trans. on Very Large Scale Integration Systems*, vol.15, no.3, pp.328-337, Mar. 2007.

[7] M. Wenk, M. Zellweger, A. Burg, N. Felber, and W. Fichtner, "K-best MIMO detection VLSI architectures achieving up to 424Mbps," *in Proc. of International Symposium on Circuits and Systems*, pp.1151-1154, May 2006.

[8] K. E. Batcher, "Sorting networks and their applications," *in Proc. of the AFIPS Spring Joint Computing Conference*, vol.32, pp.307-314, 1968.

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

[10] D. Seethaler and H. Bolcskei, "Infinity-norm sphere-decoding," *in Proc. of International Symposium on Information Theory*, pp.2002-2006, July 2008.

[11] C. P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems," *Math. Programming*, vol.66, no.2, pp.181-191, Sep. 1994.

[12] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection," *IEEE Journal of Selected Areas in Communications*, vol.24, no.3, pp.491-503, March 2006.