A threshold-based algorithm and VLSI architecture of a K-best Lattice Decoder for MIMO Systems

Jin Jie, Chi-ying Tsui, Wai-Ho Mow

Department of Electrical and Electronic Engineering

The Hong Kong University of Science and Technology

Clear Water Bay, Kowloon, Hong Kong

eetsui@ee.ust.hk

Abstract- **Lattice decoding algorithms have been shown to have the similar performance as the optimal maximum likelihood decoder for MIMO wireless systems. To reduce the high complexity of the lattice decoding algorithm and to achieve a regular fixed throughput, K-best algorithm and the corresponding VLSI architectures have been proposed for the practical implementation of the lattice decoding algorithm. In this paper, we propose a threshold-based K-best algorithm that offers significant reduction in computation and thus energy consumption, while still maintaining the performance. The method is based on the efficient pruning of the candidates in each dimension of the search tree. At the same time the throughputs of different VLSI implementations are studied and a high-throughput VLSI architecture is proposed. We show that by properly scheduling the hardware, optimal throughput can be achieved. Experimental results show that more than 40% of the computation can be reduced when the threshold-based K-best algorithm is used comparing with the conventional K-best algorithm. Also a VLSI implementation based on 0.25 μm technology that can achieve a throughput of over 50mb/s is presented.**

I. INTRODUCTION

In the past few years, significant amount of activities in both communication and VSLI research communities have been focused on how to obtain extraordinary spectral efficiency and achieve high-speed wireless data transmission. In [1], it has been shown that multi-input multi-output (MIMO) wireless communication systems are capable of transmitting data at potentially very high data rates. However, the optimal Maximum-likelihood (ML) decoder is infeasible for the MIMO system, especially when a system uses a large number of antennas together with higher modulation constellations. Recently, lattice decoding algorithm has been proposed for MIMO systems [2]. It has been shown that the performance of lattice decoder is approaching that of the ML decoder while a lower complexity is needed.

Two kinds of lattice decoder are proposed to achieve the ML Performance with lower complexity. They are the Fincke-Phost algorithm proposed in [2] (we denote it as SD algorithm) and the Schnorr-Euchner algorithm proposed in [3] (we denote it as SE algorithm). The lattice decoder can be interpreted as a tree search approach of which in the worst case the complexity is exponential. The complexity can be reduced with efficient pruning. The original depth first search decoding has the disadvantage that the computation requirement varies with the input signal and hence the decoding throughput is also varying, which is not desirable for real time signal detection.

Recently, a K-Best algorithm was proposed [4]. It uses breadth-first search instead of depth-first search. The best K candidates are kept at each search level. It is shown that the performance is very close to the optimal if K is sufficiently large. The K-Best algorithm requires less amount of computation as compared to the conventional lattice decoder and can be easily implemented in a pipelined fashion and has a fixed throughput. Corresponding VLSI architectures for the K-best algorithm based on SD and SE algorithms have also been proposed [5, 6].

In this paper, we propose a threshold-based K-best algorithm that adjusts the thresholds in each dimension to reduce the searching space for the closest point to a region much smaller than the conventional K-best algorithm. This leads to a reduction in computation complexity and hence a lower power consumption. By choosing the right threshold values at each level, the performance degradation due to the pruning is minimized. We show that in the interested SNR region, more than 40% computational reduction can be obtained while maintaining the BER performance. At the same time, a VLSI architecture is proposed which employs a proper scheduling of the hardware It is shown that the optimal throughput of the K-best algorithm can be achieved by using this architecture.

II. SYSTEM MODEL AND NOTATION

In this paper, it is assumed that the un-coded MIMO system has $\tilde{M}$ transmit antennas and $\tilde{N}$ receive antennas. The channel is assumed to be a Rayleigh frequency non-selective fading channel and it is quasi-static. The corresponding complex-valued input and output relation is

$$\tilde{\mathbf{x}} = \tilde{\mathbf{H}}\tilde{\mathbf{u}} + \tilde{\mathbf{n}} \tag{1}$$

where $\tilde{\mathbf{H}}$ is the $\tilde{N}\times\tilde{M}$ complex channel matrix, $\tilde{\mathbf{u}}$ is the $\tilde{M}\times 1$ transmitted symbol vector, $\tilde{\mathbf{x}}$ is the $\tilde{N}\times 1$ received vector, and $\tilde{\mathbf{n}}$ is the $\tilde{N}\times 1$ additive white Gaussian noise vector.

As shown in [2], the complex equation is converted to its real representation **x=Hu+n**, as follows:

$$\begin{bmatrix}\Re(\tilde{\mathbf{x}})\\ \Im(\tilde{\mathbf{x}})\end{bmatrix} = \begin{bmatrix}\Re(\tilde{\mathbf{H}}) & -\Im(\tilde{\mathbf{H}})\\ \Im(\tilde{\mathbf{H}}) & \Re(\tilde{\mathbf{H}})\end{bmatrix}\begin{bmatrix}\Re(\tilde{\mathbf{u}})\\ \Im(\tilde{\mathbf{u}})\end{bmatrix} + \begin{bmatrix}\Re(\tilde{\mathbf{n}})\\ \Im(\tilde{\mathbf{n}})\end{bmatrix} \tag{2}$$

Let $N = 2\tilde{N}$ and $M = 2\tilde{M}$, then **x** and **n** are *Nx1* vectors , **u** is a *Mx1* vector and H is a *NxM* real matrix.

In this paper, we assume the constellation used is 16-QAM. Thus for each entry (u') of $\tilde{\mathbf{u}}$, $u' = a + jb$, where a, b = ±1 or ±3. Thus, the constellation of each entry of **u** (i.e. the real part or the imaginary part of u') is $\Omega = \{-3,-1,1,3\}$.

III. THE K-BEST SD ALGORITHM

The optimal MIMO detection is to find a lattice point **U** in space Ω^M, such that its transformed point **HU**, has the minimum Euclidean distance to a noisy observation vector **X**, i.e.,

$$\hat{\mathbf{U}} = \arg \min_{\mathbf{U} \in \Omega^M} \| \mathbf{x} - \mathbf{H}\mathbf{U} \|^2 \tag{3}$$

Both SD decoder and SE decoder can be interpreted as a depth-first tree search approach with pruning, while the K-best algorithm uses the breadth-first search instead of depth-first search and limits the searching scope in each dimension.

Let $\mathbf{p}=\mathbf{H}^{-1}\mathbf{x}$, $\mathbf{e}=\mathbf{p}-\mathbf{u}$ and $\mathbf{H}^T\mathbf{H}=\mathbf{R}^T\mathbf{R}$, where **R** is an upper triangular matrix. Also let $\hat{\mathbf{P}}$ be the zero-forcing estimation of **p, Q** be an upper triangular matrix and $Q_{i,i}=R_{i,i,}^{\ 2}$, $Q_{i,j}=R_{i,j}/R_{i,I,}$ where $i=1:M ,j=i+1:M$. Equation (3) can be written as follows:

$$\begin{aligned}\hat{\mathbf{U}} &= \arg\min_{\mathbf{U}} \| \mathbf{H}(\mathbf{p}-\mathbf{U}) \|^2 = \arg\min_{\mathbf{U}} \| \mathbf{H}\mathbf{e} \|^2 \\ &= \arg\min_{\mathbf{U}} \sum_{i=1}^{M} Q_{i,i}\left(e_i + \sum_{i=i+1}^{M} Q_{i,j} e_j \right)^2 \\ &= \arg\min_{\mathbf{U}} \sum_{i=1}^{M} D_i\end{aligned}$$

where $D_i = Q_{i,i}(e_i + \sum_{i=i+1}^{M} Q_{i,j}e_j)^2$, which is the distance of ith dimension.

The search starts from the symbol *M*. In each dimension only the paths with the accumulated partial Euclidean distance less than *C* , where *C* is the Euclidean distance between the received vector and an initial estimate which is the zero-forcing vector $\hat{\mathbf{P}}$, will be kept. If the number of the paths is larger than K, then only the K-best paths will be kept.

The K-best algorithm starts with the pre-calculations which compute the decoding order and the values of $\mathbf{H}, \mathbf{p}, \hat{\mathbf{P}}, \mathbf{Q}, C$. $\mathbf{H}, \mathbf{p}, \hat{\mathbf{P}}, \mathbf{Q}$ are calculated once for every frame and *C* is calculated for every received vector. Then the decoding algorithm is executed by carrying out the following steps:

Step1. Input $\mathbf{p}, \hat{\mathbf{P}}, \mathbf{Q}, C$, and initialize:

$K=M$

$S_k=p_M$

bestdist=*C*

Step2. For $i = 1,2,\cdots,length(\mathbf{bestdist})$, where the length function returns the number of elements in (.), calculate:

$$u_{t,k} = \omega, \forall\, \omega \in \Omega$$

$$D_t=Q_{k,k}(S_{i,k}-u_{t,k})^2$$

$$newdist_t=bestdist_i - D_t$$

Step3. Let *T=length(***bestdist***)*length(*Ω*)*, sort $newdist_t$ (t=1,2,…,T) in descending order. The candidates whose $newdist_t$ value less than 0 will be discarded since the accumulated partial distance up to this candidate is then greater than *C*. Among the candidates remained, we keep the K (or T depending on which is smaller) best candidates. Then we adjust the **u,s,e** accordingly, and replace **bestdist** with **newdist**.

Step4. For *i=1,2,…,length(***bestdist***)*, calculate:

$$e_{k,i} = p_k - u_{k,i}$$

$$S_{k-1,i} = p_{k-1} + \sum_{j=k}^{M} Q_{k-1,j} e_{j,i}$$

Step5. If k≠1, then go to step 2) with k=k-1; else if the first elements of **bestdist** is larger than zero, return the first row of **u** as $\hat{\mathbf{U}}$, else return $\hat{\mathbf{P}}$ as $\hat{\mathbf{U}}$.

We denote the step2 as the T phase calculation, step3 as the sorting calculation, and step4 as the S phase calculation.

IV. THE THRESHOLD-BASED K-BEST ALGORITHM

Originally, the K-best algorithm is to find the K-best paths that meet the following threshold at every level (k>1) and choose the best path at the last level (k=1).

$d_8 \le C$

$d_8+d_7 \le C$

…

$d_8+d_7+\ldots\ d_1 \le C$

Note that at the *k* (k>1)th level, the upper bound *C* is a very loose bound. The reason is based on the following two facts:

1. The initial distance got from the zero-forcing is not accurate and always larger than the minimum Euclidean distance, especially when the channel condition is bad.

2. The total Euclidean distance is the sum of the partial distance, which is linear with N. At the *k* (k>1)th level (i.e. before the leaf node), the accumulated sum of the distance is surely less than the total distance, especially for the beginning stages.

For the K-best SD algorithm, we can modify the algorithm to start pruning the tree at the early levels, just like in [6] pruning the tree for the MLD decoding. In step 3 , instead of comparing the $newdist_t$ with 0, we check the candidate with the following threshold value that is varying with level to see whether we should discard the candidate "$newdist_t$>C-α_i*i*$C/8$", where *i=9-k*.

The threshold value is adjusted with different levels. To minimize the effect on the bit-error rate, it is expected α_k is descending. From the simulation results, we use the threshold set $\boldsymbol{\alpha}_1$=[1.6,1.5,1.4,1.3,1.2,1.1,1.0,1.0]. In the extreme case, we can set $\boldsymbol{\alpha}_2$=[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0], which means the partial distance threshold value increases linearly with the level increasing. For the K-best SE algorithm in [6], although the initial distance is empirically set from the simulation, we still can set different bounds at different levels. The pruning scheme is the same as the K-best SD algorithm.

We simulate the performance and the computational reduction of a 4-transmit and 4-receive antennas system. 1000 packets of 400 uncoded 16-QAM symbols are transmitted with 100 symbols for each antenna. The average energy per symbol is fixed to 10, and the variance of σ^2 of the AWGN is adjusted by

$$\sigma^2=(1/2)*\tilde{N}*(E_s/\log_2\Omega)*10^{-SNR/10} \quad (4)$$

Figure 1 shows the BER performance of the algorithm. It can be seen that the BER of the K-best algorithm with our pruning scheme using the first set of coefficients is almost the same as the original K-best algorithm. Even if we use the second set of metric, the BER is only slightly degraded. This is due to the upper bound adopted by the original K-best algorithm is very loose. The search trees pruned by our scheme rarely contain the path with the smallest Euclidean distance.

Figure 2 shows the reduction in computation operation. It can be seen that in the SNR of interests (SNR≥20dB), using our pruning scheme, over 40% computational reduction can be achieved. The pruning scheme becomes more efficiently, when the SNR is higher. The power consumption is proportional to the computational complexity. Therefore, using our pruning scheme, more than 40% power reduction can be achieved.

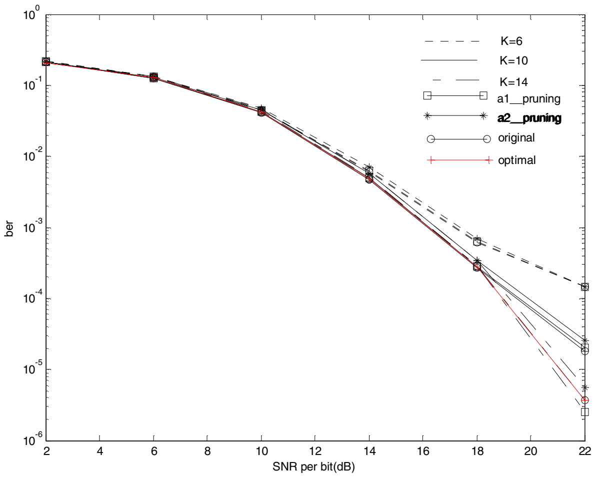

Figure 1. The performance comparison

V. ADAPTABLE K-BEST ALGORITHM VLSI IMPLEMETNATION ISSUES

The K-best algorithms can be easily implemented in VLSI using the pipelined architecture. Because of the regular computations, it is flexible and scalable. In this section, the VLSI implementation issues for a 16 QAM system with 4 transmit/receive antennas are studied.

At the algorithm level, some modification can be made to reduce the computation complexity of the K-best SD algorithm. As in [6], we can modify the T-phase calculation in step2 with $D_t=(R_{k,k}s_{i,k}-\tilde{u}_{t,k})^2$, where $\tilde{u}_{t,k}=R_{k,k}u_{t,k}$. It can be calculated in the pre-calculations and shared in one iteration. By this modification, the complexity can be reduced, but it requires extra memory (256 bits if 16 bits resolution is used) and increases the workload of the preprocessing units. Considering the overhead, we will not use this reduction in the following analysis.

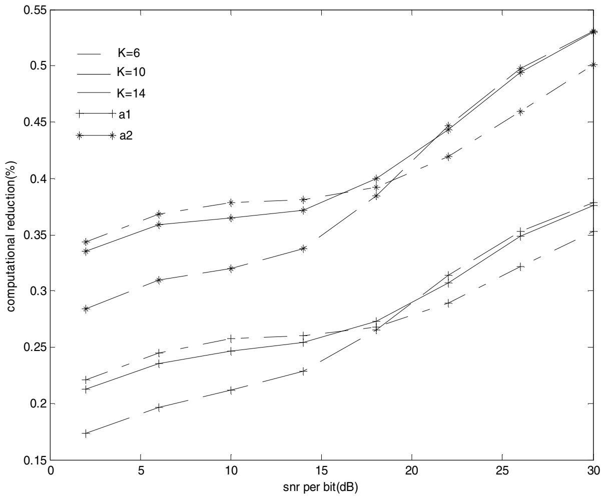

Figure 2. The computational reduction comparison

When we re-examine the execution of the K-best algorithm, it can be seen that the execution involves three types of calculations:

1) The T phase calculation in step 2: In the worst case, it involves 8K subtractions, 4K square operations and 4K multiplications.

2) The sorting calculation in step 3. Basically, the operation is “compare and select“. The number of calculations is linear with K. In VLSI implementation, this computation can be merged with the T-phase calculation [5].

3) The S phase calculation in step 4. The number of calculations is different from stage to stage. In the last stage (i.e. k=1), we do not need the S-phase calculation since the final result is obtained after the T-phase calculation. Therefore the largest number of calculations requirement is in the 7th (k=2) stage. In the worst case, it involves K subtractions, 7K multiplications and 7K additions.

One of the most important factors with the implementation is the throughput. When a pipelined architecture is adopted, the throughput is limited by the maximum clock cycles requirements in the T and S phase calculations. In [5], in order to save hardware, the T phase and S phase calculations are processed by the same processing element (PE) which includes one adder and one multiplier. The whole architecture is partitioned into 8 stages and each stage has it corresponding PE. In the same stage, the S phase calculations must wait for the T phase calculation. The maximum clock cycle requirement is in the 7th (k=2) stage. . In 7th stage, the S phase calculation requires 8K clock cycles to finish the addition and the subtraction. Together with 8K clock cycles required in the T phase calculation, the total clock cycles requirement is 16K. Thus, the throughput is *16/16Kt* b/s, where *t* is the clock period. To improve the throughput, we can decouple the T-phase and S-phase calculation by adding parallel hardware.

By adding more hardware, two adders and two multipliers for the T phase calculation, two adders (one of them executes the subtraction) and one multiplier for the S phase calculation, each phase of the calculation can be calculated by one processing element, and the whole calculation can be pipelined in the way as shown in Fig. 3.In this case, the maximum clock cycle requirement is to calculate $S_{1,i}=p_1+Q_{1,2}e_{2,i}+\ldots+Q_{1,8}e_{8,i}$ for i=1:K, which is at S2(k=2) stage and the number of clock cycles is 7K. Thus, the throughput is *16/7Kt* b/s.

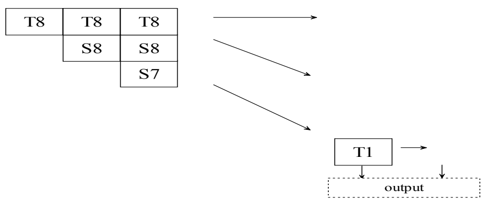

Figure 3. The pipelined calculation.

Reordering the calculation in the S phase will further reduce the clock cycles. In step 1, instead of just calculating S_k, (where k = 8) by $S_k=p_M$, we calculate all the S_i values (i from 1 to 8) with **S**=**p**. In step3 we adjust the values of **u,s** accordingly after the sorting. In step4 we modify the S-phase calculation by the following:

$$e_{k,i}=p_k-u_{k,i}, \quad S_{j,i}=S_{j,i}+Q_{j,k}e_{k,I} \quad j=1:k-1$$

Here in the stage k, $e_{k,i}$ is ready and it can be used for the partial calculation of $S_{k,i}$ where j is from 1 to k-1. So some fo the computations at the later stages can be moved forward to the earlier stages. For $K \geq 5$, the maximum clock cycle requirement is in S7(k=7) stage calculation which calculates $S_{j,i}=S_{j,i}+Q_{j,7}e_{7,i}$ for j=1:6 and i=1:K and the maximum clock cycles are 6K. Thus, the throughput is *16/6Kt* b/s. The K-best in [6] is this type of implementation.

Note that the number of calculations in S phase is different from stage to stage. Further improvement can be achieved by properly scheduling the S phase calculations. By storing $e_{8,i}$we can move the calculation $S_{j,i}=S_{j,i}+Q_{j,8}e_{8,i}$, for j=1:3, i=1:K, in the S8 stage to the S2 stage. In the same way, by storing $e_{k,i}$ we can move the calculation $S_{j,i}=S_{j,i}+Q_{j,k}e_{k,i}$, for j=1:2, i=1:K, k=6:7, in the S7 and S6 stages to the S3 and S4 stages, respectively. Now the maximum clock cycles required in the S-phase is equal to 4K. By evenly distributing the calculations, the optimal throughput can be obtained and it is equal to *16/4Kt* b/s.

Combining the new pruning scheme and the optimal scheduling of the S phase calculations, we propose a new K-best architecture. Fig. 4 gives the whole pipelined architecture. Fig. 5 shows the second pipelined stage. The other pipelined stages are almost the same, except that one multiplexer is added to the k=2, 3 and 4 stages to execute the calculations delayed from the k=8, 7 and 6 stages. We show the throughputs of the different implementations in Fig 6. We have designed the decoder for a 16-QAM system with 4 transmit/receive antennas in a 0.25µm technology and the core area is 3.64mm x 3.68mm. The post-layout critical path delay of a PE is 10ns. The throughput shown in Fig. 6 is based on this delay value. From Fig. 6, it can be shown that for a given K, the throughput of the proposed K-best architecture outperforms the other K-best architectures presented in [5, 6].

VI. CONCLUSION

A new pruning scheme for K-best algorithm is proposed. Comparing with the conventional K-best algorithm, over 40% power reduction can be achieved. Our pruning scheme can be easily implemented by adding a comparator before the sorting elements to the conventional K-best architecture. The throughput of various implementations was also studied. We show that optimal throughput can be achieved by properly scheduling the computation.

REFERENCES

[1] I. E. Telatar, ”Capacity of multi-antenna gaussian channels,” *Eur. Trans. Telecom.*, vol.10, pp.585-595, Nov. 1999

[2] O. Damen, A. Chkief, and J. C. Belfiore, ”Lattice code decoder for space-time codes,” *IEEE Communications letters*, vol.4,no. 5, pp.161-163, May 2000.

[3] E. Argell, E. Eriksson, A. Vardy, and K. Zeger, “Closest point search in lattices,” *IEEE Transactions on Information Theory*, vol. 48, no. 8, pp. 2201-2214, August 2002.

[4] Kwan Wai Wong, Chi-Ying Tsui, Roger S Cheng, Wai Ho Mow, “Reduced-complexity Maximum Likelihood lattice decoder for MIMO channels,”, in the Proceedings of the 7th Asia-Pacific Conference on Communications, pp. 213-216, 2001

[5] K. -W. Wang, C. -Y. Tsui, R. S. -K. Cheng, and W. -H. Mow, “ A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels”, *IEEE ISCAS*, vol.3, pp. 273-276, May 2002.

[6] Zhan guo and Peter, Nilsson, “A VLSI architecture of Schnorr-Euchner Decoder for MIMO channels”, *IEEE CAS Symp. On Emerging Technology: Mobile and Wireless Comm*, vol.1, pp. 65-68, May 2004.

[7] Gowaikar, R. and Hassibi, B., “Efficiently statistical pruning for maximum likelihood decoding”, *ICASSP On Acoustics, Speech, and Signal Processing*, vol.5, pp. 49-52, April 2003.v

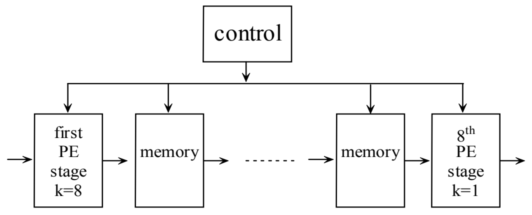

Figure 4. The pipilined architecture of K-best algorithm

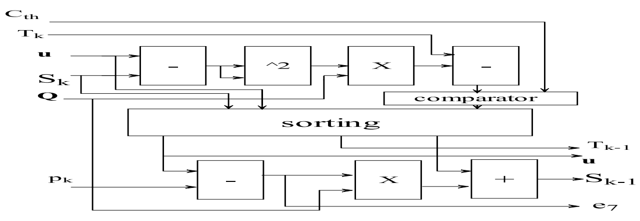

Figure 5. The second (k=7) pipelined stage of K-best

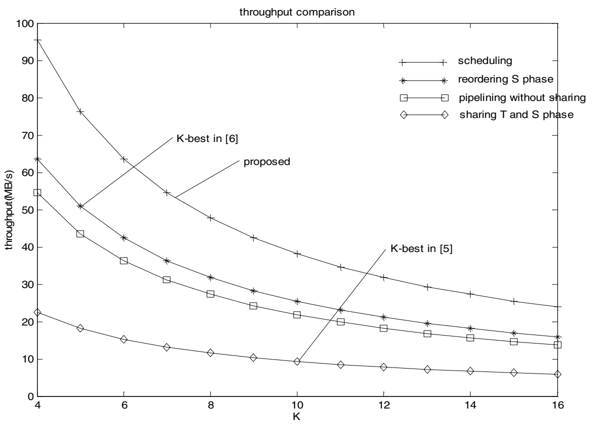

Figure 6. The throughput of different implementations