Reduced QRD-M Detector in MIMO-OFDM Systems With Partial and Embedded Sorting

Yuanbin Guo
Nokia Research Center
Irving, Tx 75039
Email: Yuanbin.Guo@nokia.com

Dennis McCain
Nokia Research Center
Irving, Tx, 77039
Email: Dennis.McCain@nokia.com

Abstract—In this paper, we present a reduced QRD-M matrix symbol detector in MIMO-OFDM systems. The QRD-M algorithm first decomposes the MIMO channel matrix into upper triangular matrix and applies a limited tree search to approximate the maximum-likelihood detector. The metric update is reduced from $\mathcal{O}(4.5MC)$ to $\mathcal{O}(1.5MC)$ by extracting the commonality. We then propose a partial quick-sort procedure and an embedded insert sort to achieve almost linear sorting. In the second part, we present efficient VLSI architectures utilizing the parallelism between subcarriers and design the pipelining in the multi-stage MIMO processing. The real-time architecture is implemented in a FPGA-based hardware accelerator with compact form factor, which achieves up to $100\times$ speedup in the simulation time.

I. INTRODUCTION

MIMO (Multiple-Input-Multiple-Output) technology [1] [2] using multiple antennas at both the transmitter and receiver sides has emerged as a significant breakthrough to increase the system throughput dramatically. Recently, the combination with OFDM (Orthogonal Frequency Division Multiplexing) technology leads to the MIMO-OFDM technology [3] as a strong candidate for the 4G wireless systems. It is known that the optimal Maximum Likelihood (ML) detector [5] leads to much better performance than the conventional VBLAST algorithm for symbol detection. However, the complexity increases exponentially with the number of antennas and symbol alphabet, which is prohibitively high for practical implementation.

To achieve a good tradeoff between performance and complexity, a suboptimal QRD-M algorithm was proposed in [4] to approximate the ML detector. The QR-decomposition [6] reduces the channel matrices for N_T transmit and N_R receive antennas to upper triangular matrices. The M-search algorithm then limits the tree search to the M smallest branches in the metric computation. The complexity is significantly reduced compared with the full tree search of the ML detector. It was shown that the performance approaches the optimal detector with increasing M , while the VBLAST algorithm could be considered as a special case when $M = 1$.

Despite of the significantly reduced complexity, the QRD-M algorithm is still the bottleneck in the receiver design, especially for high-order modulation, high MIMO antenna configuration and large M . It is shown that in a matlab MIMO-OFDM simulation chain, the QRD-M algorithm can occupy

more than 99% simulation time even with the help of C-MEX file implementation. It can take days or even weeks to generate one performance point. This not only slows the research activity severely, but also limits the practicability of the QRD-M algorithm in real-time implementation. To shorten the simulation time and facilitates the commercialization, a hardware accelerator platform with compact form factor was proposed in [7] based on Xilinx FPGAs to achieve functional verification of the fixed-point hardware design. It also allows rapid prototyping of the VLSI architectures for proof of concept in the future real-time 4G systems. However, the tree search structure is not quite suitable for VLSI implementation because of the intensive memory operations with variable latency, especially for long sequences. Extensive algorithmic optimizations are required to achieve an efficient VLSI architecture.

In this paper, we propose algorithmic optimizations to reduce the complexity of the conventional QRD-M algorithm. We first extract the commonality in the metric update process to reduce the redundancy in numerical operations. The numerical complexity of the M-algorithm is reduced from $\mathcal{O}(4.5MC)$ to $\mathcal{O}(1.5MC)$ in terms of complex multiplications, where C is the size of the symbol alphabet and M is number of the survivor branches for the limited tree search. In the tree search, the selection of sorting method depends on the sequence length and stochastic order of the sequence to be sorted. Theoretically, quick-sort is the fastest sorting function with $\mathcal{O}(N \log_2 N)$ complexity, where $N = MC$. For large M and C , the sorting instead of the metric computation becomes the bottleneck in latency. The extensive memory access may stall the pipelining and introduce variable latency. We show that in the problem of M-search [8], there is no need to do a complete sorting of the $N = MC$ sequence. Only the smallest M survivors need to be sorted. We propose a modified partial quick-sort procedure and achieve almost linear sorting latency. This is suitable for high-order processing which requires extensive memory operations. For the relatively small M and C , we study the stochastic order of the metric sequence and propose an embedded insert sort in the metric computation loop which requires small storage buffers.

The efficient VLSI architectures are explored by a Catapult C high-level synthesis design methodology [9]. For a $C = 64$ (64-QAM) and $M = 64$ system, speedup of $100\times$ is

Fig. 1. System Model of the MIMO-OFDM Using Spatial Multiplexing.

observed with 33 MHz FPGA clock rate competing with the 1.5 GHz Pentium-4 clock rate. The current architecture provides a reference for the real-time prototyping of the future 4G systems.

II. SYSTEM MODEL

We consider the equivalent baseband signal model for the MIMO OFDM system with N_T transmit and N_R receive antennas. The system model is shown in Fig. 1. We assume that there are N_F subcarriers. At the transmitter, the high data rate bit stream is broken down into N_T lower rate substreams by spatial multiplexing. At the p^{th} transmit antenna, the multiple bit substreams are modulated by constellation mappers to some BPSK, QPSK, 16-QAM or 64-QAM symbols. $d_p(\kappa * N_F + k)$ is the k^{th} frequency domain subcarrier during the $[\kappa T_d^g, (\kappa + 1)T_d^g]$ OFDM symbol epoch and T_d^g is the symbol duration including the guard time. For simplicity in the notation, we focus on the κ^{th} symbol and omit the symbol index κ . During this symbol epoch, the symbols are modulated by N_T parallel N_F -point IFFTs to generate the n^{th} time domain OFDM subcarrier as $s_p(n) = \frac{1}{\sqrt{N_F}} \sum_{k=0}^{N_F-1} d_p(k) e^{j2\pi kn/N_F}$. After the insertion of the cyclic prefix, the time domain symbol is given by $\tilde{s}_p(n)$. N_g is the number of subcarriers in the guard interval. $T_d^g = (N_F + N_g)T_s$ is the symbol period where T_s is the sampling period of the IFFT. Thus the analog signal transmitted at the p^{th} transmit antenna is given by $\tilde{s}_p(t) = \sum_n \tilde{s}_p(n) p(t - T_g - nT_d^g)$ where $p(t)$ is the pulse shape function with support $[0, T_s]$ and $T_g = N_g T_s$ is the guard time. By passing the signal through a multipath fading channel and after sampling at the same rate, the received signal at the q^{th} receive antenna is given by

$$r_q(n) = \frac{1}{\sqrt{N_F}} \sum_{p=1}^{N_T} \sum_{l=0}^{L_{p,q}} h_{p,q}(l) \sum_{k=0}^{N_F-1} d_p(k) e^{j2\pi kn/N_F} + v_q(n)$$

where $L_{p,q}$ is the channel delay spread between the p^{th} transmit and the q^{th} receive antennas. $v_q(n)$ is the additive Gaussian noise to the q^{th} antenna. The MIMO channel is characterized by a matrix given by the Tapped Delay Line

(TDL) model as $h_{p,q}(t) = \sum_{l=0}^{L_{p,q}} h_{p,q}(l) \delta(t - \tau_{p,q,l})$ where $\delta(t)$ is the Kronecker Delta function. $h_{p,q}(l) \in \mathbb{C}$ and $\tau_{p,q,l}$ are the amplitude and delay of the l^{th} path channel coefficient respectively. The cyclic prefix guard time satisfies $T_g > \max(\tau_{p,q,l})$ to eliminate the Inter-Symbol Interference (ISI). Thus, OFDM converts the multipath frequency-selective fading channel into flat fading channel and simplify the channel estimation.

We collect the N_F samples in a vector as $\mathbf{r}_q = [r_q(0), r_q(1), \dots, r_q(N_F - 1)]^T$ and define the DFT phase coefficients as $\Phi_k = 1/\sqrt{N_F} [1, e^{j2\pi k/N_F}, e^{j2\pi 2k/N_F}, \dots, e^{j2\pi (N_F-1)/N_F}]^T$. The DFT matrix is formed as $\Phi = [\Phi_0, \Phi_1, \dots, \Phi_{N_F-1}] \in \mathbb{C}^{N_F \times N_F}$. The received signal is then given in a vector form as

$$\mathbf{r}_q = \sum_{p=1}^{N_T} \Phi \mathbf{H}_{p,q} \mathbf{d}_p + \mathbf{v}_q \quad (1)$$

where $\mathbf{v}_q$ is the noise vector, $\mathbf{H}_{p,q}$ is the frequency domain channel matrix formed from FFT result of the channel coefficients.

$$\begin{aligned} \mathbf{H}_{p,q} &= \text{diag}\{H_{p,q}(0), H_{p,q}(1), \dots, H_{p,q}(N_F - 1)\}, \\ H_{p,q}(k) &= \Phi_k^H \cdot \mathbf{h}_{p,q}, \\ \mathbf{h}_{p,q} &= [h_{p,q}(0), h_{p,q}(1), \dots, h_{p,q}(N_F - 1)]^T. \end{aligned} \quad (2)$$

$\mathbf{d}_p = [d_p(0), d_p(1), \dots, d_p(N_F - 1)]^T$ is the transmitted frequency domain symbol vector. At each of the q^{th} receive antenna, a N_F -point FFT is operated on the received signal to demodulate the frequency domain symbols as

$$\mathbf{y}_q = \Phi^H \mathbf{r}_q = \sum_{p=1}^{N_T} \mathbf{H}_{p,q} \mathbf{d}_p + \mathbf{z}^q \quad (3)$$

where $\mathbf{z}^q = \Phi^H \mathbf{v}_q$ is the noise vector in the frequency domain.

III. REDUCED PARTIAL QRD-M DETECTOR

A. Joint Detector

The goal of the receiver is to detect the $\mathbf{d}_p$ effectively from the received signal and estimated channel coefficients $\hat{\mathbf{H}}_{p,q}$. It is shown [4] that the ML data detector is given by performing one-step prediction on the channel using Kalman filter as

$$\hat{\mathbf{d}}_{p,ML} = \arg \min_{\mathbf{d}_p \in \{S\}} \|\mathbf{y}_q - \hat{\mathbf{H}}_{p,q}(n|n-1) \mathbf{d}_p\|^2 \quad (4)$$

where $\hat{\mathbf{H}}_{p,q}(n|n-1)$ is the one-step Kalman filter prediction of the effective MIMO channel frequency response matrix and S is the symbol alphabet. It is shown that the symbol detection is separable according to the subcarriers, i.e., the components of the N_F subcarriers are independent. Thus, this leads to the subcarrier-independent symbol detection as

$$\mathbf{d}_{ML}^k = \arg \min_{\mathbf{d}^k \in \{S\}^{N_T}} \|\mathbf{y}^k - \hat{\mathbf{H}}^k(n|n-1) \mathbf{d}^k\|^2, \quad (5)$$

where $\mathbf{y}^k = [y_1^k, y_2^k, \dots, y_{N_R}^k]^T$ is the k^{th} subcarrier of all the receive antennas.

$$\mathbf{H}^k = \begin{pmatrix} H_{1,1}^k & H_{2,1}^k & \dots & H_{N_T,1}^k \\ H_{1,2}^k & H_{2,2}^k & \dots & H_{N_T,2}^k \\ \vdots & \vdots & \ddots & \vdots \\ H_{1,N_R}^k & H_{2,N_R}^k & \dots & H_{N_T,N_R}^k \end{pmatrix} \quad (6)$$

is the channel matrix of the k^{th} subcarrier. $\mathbf{d}_k = [d_1^k, d_2^k, \dots, d_{N_T}^k]^T$ is the transmitted symbol of the k^{th} subcarrier for all the transmit antennas. This detector has prohibitively high complexity which grows exponentially with the number of transmit antennas and the cardinality of the subcarrier modulation. This makes the detector unrealistic for practical real-time implementation.

To reduce the complexity, a suboptimal M-algorithm for joint detection has been proposed in [4]. First the channel estimates from all the transmit antennas are sorted using the second-order statistics of the estimated powers to make $\hat{P}_2^{(n_1)} \leq \hat{P}_2^{(n_2)} \leq \dots \leq \hat{P}_2^{(n_T)}$. The re-ordered channel vector is given by $\hat{\mathbf{H}}^k = [\hat{\mathbf{H}}_{(n_1)}^k, \hat{\mathbf{H}}_{(n_2)}^k, \dots, \hat{\mathbf{H}}_{(n_T)}^k]$. The data vector $\mathbf{d}^k$ is also re-ordered accordingly in terms of the corresponding powers as $[d_{(n_1)}^k, d_{(n_2)}^k, \dots, d_{(n_T)}^k]^T$. Then the QR decomposition [6] algorithm is applied to the estimated channel matrix for each subcarrier as $\mathbf{Q}_k^H \hat{\mathbf{H}}^k = \mathbf{R}_k$ where $\mathbf{Q}_k$ is the unitary matrix and $\mathbf{R}_k$ is an upper triangular matrix. The FFT output symbols y_k are pre-multiplied by the $\mathbf{Q}_k^H$ matrix to form a new receive signal as

$$\Upsilon_k = \mathbf{Q}_k^H y_k = \mathbf{R}_k \mathbf{d}_k + \mathbf{w}_k \quad (7)$$

where $\mathbf{w}_k = \mathbf{Q}_k^H \mathbf{z}_k$ is the new noise vector. To reduce the complexity of the ML detector, the suboptimal M-algorithm is proposed to approximate the ML decisions on $\mathbf{d}_k$ using limited tree search. The cost function is determined in terms of states and branch metrics as

$$\hat{\mathbf{d}}_{ML}^k = \arg \min_{\mathbf{d}^k \in \{\mathcal{S}\}^{N_T}} \sum_{p=1}^{N_T} m_p((\Upsilon_k); s_{(p)}) \quad (8)$$

where the metric and the state are defined respectively by

$$\begin{aligned} m_p((\Upsilon_k); s_{(p)}) &= \|(\Upsilon_k)_{N_T-p+1} - (\mathbf{R}_k)_{N_T-p+1, N_T-p+1} \\ &\quad \cdot d_{N_T-p+1}^k - \sum_{n_t=N_T-p+2}^{N_T} (\mathbf{R}_k)_{N_T-p+1, n_t} d_{n_t}^k \|^2, \\ s_p &= \{d_{N_T-p+1}^k, d_{N_T-p+2}^k, \dots, d_{N_T}^k\}. \end{aligned} \quad (9)$$

The ML detector is equivalent to a tree search beginning at level (1) and ending at level (N_T), which has a prohibitive complexity at the final stage as $\mathcal{O}(|\mathcal{S}|^{N_T})$. The M-algorithm only retains the paths through the tree with the M smallest aggregate metrics. This forms a limited tree search algorithm which consists of both the metric update and the sorting procedure.

Fig. 2. The block diagram of the limited-tree search in QRD-M algorithm.

B. Reduced Complexity QRD-M Detector

Although the QRD-M algorithm already reduces the complexity of the ML detector, it is still too complicated for real-time implementation. In this section, we first analyze the commonality in the metric update procedure and then we propose a reduced complexity scheme. In this section, we briefly derive the reduced scheme.

The QRD-M procedure is depicted in Fig. 2 for an example with QPSK modulation and N_T transmit antennas. The data $d_{N_T}^k$ associated with the strongest channel is allocated as the root node at stage 1. At this stage there are C candidate points to detect, where C is the size of the symbol alphabet $\mathcal{S}$ for different modulation scheme. For a M limited tree search, the metric at each stage is sorted and only the M smallest branches are maintained as the survivor branches. Then for the next stage, the derived C metrics from each of the M survivor branches are computed and added to the aggregate metrics of the previous survivor branches to form a $M * C$ metric sequence. This sequence is sorted again and only the M smallest branches survive till the next stage. Thus for each stage, there are at most $M * C$ metric branches to compute instead of C^p branches at the p^{th} stage for the full tree search in the conventional ML detector.

The upper triangular channel matrix after the QR decomposition has the form of

$$\mathbf{R}_k = \begin{pmatrix} R_k(1,1) & R_k(1,2) & \dots & R_k(1,N_T) \\ 0 & R_k(2,2) & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & R_k(N_T,N_T) \end{pmatrix}. \quad (10)$$

We define the re-ordered new receive signal vector as $\Upsilon_k = [\Upsilon_k(1), \dots, \Upsilon_k(N_T)]^T$ and the symbol alphabet vector with length C as $\mathcal{S} = [\mathcal{S}(0), \dots, \mathcal{S}(C-1)]^T$. Let n_c as the constellation point index in the symbol al-

phabet. Specifically, the temporary metric vector $\mathcal{M}_p = [\mathcal{M}_p(0), \dots, \mathcal{M}_p(C), \dots, \mathcal{M}_p(MC-1)]^T$ of length MC at stage p is calculated in terms of the distance of the received signal to the symbol alphabets with channel coefficients as

$$\begin{aligned} \mathcal{M}_p^k(m * C + n_c) &= |\Upsilon_k(N_T - p + 1) \\ &- R_k(N_T - p + 1, N_T - p + 1) \cdot \mathcal{S}(n_c) \\ &- \sum_{l=N_T-p+2}^{N_T} R_k(N_T - p + 1, l) \hat{d}_l^k(m)|^2. \end{aligned} \quad (11)$$

where $\hat{d}_l^k(m)$ is the temporary hard-decision symbols for the m^{th} survivor branch at the l^{th} stage. The aggregate metric is generated by adding the aggregate metric at the $(p-1)^{th}$ stage as $\mathcal{M}_p^k(m * C + n_c) + \mathcal{M}_{p-1}^k(m)$, where $\mathcal{M}_{p-1}^k(m)$ is the sorted survivor aggregate metrics at the $(p-1)^{th}$ stage. Then this temporary aggregate metric sequence is sorted in ascending order to find both the order indices $\mathbf{I}_p^k = [I_p^k(0), I_p^k(1), \dots, I_p^k(MC)]$ and the sorted metric sequence as

$$(\widetilde{\mathcal{M}}_p^k, \mathbf{I}_p^k) = \text{sort}\{\mathcal{M}_p^k\}. \quad (12)$$

so that $\widetilde{\mathcal{M}}_p(0) \leq \widetilde{\mathcal{M}}_p(1) \leq \dots \leq \widetilde{\mathcal{M}}_p(C) \leq \dots$ and $\widetilde{\mathcal{M}}_p(i) = \mathcal{M}_p(I_p^k(i))$. The M preliminary hard-decision symbols are picked up from the symbol alphabet as $\hat{d}_p^k(0 : M-1) = \mathcal{S}\{I_p^k(0 : M-1) \% C\}$ and the symbols of the previous stages are re-arranged accordingly, where $\%$ denotes the modulo operation. This procedure is repeated until the last antenna is detected.

The complexity are dominated by the metric update procedure and the sorting function. Without exploring the commonality, the number of numerical complex multiplications is $(p+0.5)MC$ for the p^{th} stage. However, we notice that there is no survival branches from the previous stage for the first stage. Thus, the complexity from the stage 1 is $1.5C$. Moreover, the second term and the third term in (11) are not variable to either the branch index m or the symbol alphabet index n_c . We extract the commonality and define the following terms independent of the metric index as

$$D_p^k(n_c) = R_k(N_T - p + 1, N_T - p + 1) \cdot \mathcal{S}(n_c); \quad (13)$$

$$D_{p,sum}^k(m) = \sum_{l=N_T-p+2}^{N_T} R_k(N_T - p + 1, l) \hat{d}_l^k(m). \quad (14)$$

These two terms are calculated separately in advance and stored in buffers before computing the final metric power. Then for the $M * C$ loop, there are only some buffer loading and a $|\cdot|^2$ operation. In this way, we reduce the number of complex multiplications to $\mathcal{O}(M(p-1) + C + 0.5MC)$ for the p^{th} stage at the cost of two small buffers.

C. Partial Quick-Sort

Although the number of complex multiplications is an important complexity indicator because it determines the number of multipliers in a VLSI design, the real-time latency bottleneck is the sorting function. This is because the metric

computation can be pipelined in the VLSI architecture with a regular structure, but the sorting function involves extensive memory access, conditional branching and element swapping etc depending on the ordering feature of the input sequence.

There are different choices of sorting functions with trade-offs between latency and complexity. The length of the sequence is an important factor in selecting the sorting procedure. The simple bubble-sort in general has $\mathcal{O}((MC)^2)$ operations for a sequence of length MC , which is not suitable for long sequence. Theoretically, the fastest sort function has the complexity at the order of $\mathcal{O}(MC * \log_2(MC))$. Variations exist with different hidden constants in the complexity order. Since in general the quick-sort is the fastest sorting function, we propose the quick-sort function for implementation. Even though, the complexity of the full sorting is too high. For example, for a 64-QAM with $M = 64$, the sequence length is 4096. Then there are at least 40152 operations. If the sequence needs to be stored in block memory, this means at least this many cycles in hardware latency without counting the swapping, branching overheads. This results in $500\mu s$ for a single subcarrier and one antenna assuming 100 MHz clock rate, which is very challenging to meet the real-time requirement.

We make the following observations to optimize the algorithm. First, for the first stage, the sequence length is C instead of MC . Second, for the last stage, there is no need to sort the sequence as the central antennas. We only need to find the minimal value and index from the metric sequence. This procedure can be easily embedded into the metric update loop and pipelined. The bottleneck comes from the sorting of the central antennas. However, we note that because we only retain the M smallest survivor branches, we do not care the order of the other sequences above the M smallest metric. So only the M smallest metrics from the MC metric sequence need to be sorted. Using this observation, we modified the standard “quick-sort” procedure to the so-called “partial quick-sort” architecture.

To support the high modulation order and large M , the updated metrics are first stored in an input buffer memory block. Once the metric update process is ready, it sends a “start” signal to the *PQSort* procedure. The partial quick sort has the same concept as the conventional quick sort based on “partition-exchange” method. First a “partitioning element” a is selected from the subsequence. A pair of pointers il and ir are defined to set the boundary in terms of “left” and “right” sides of the subsequence. A stack *istack* with the length of *MSTACK* is allocated to store the intermediate il and ir pointers of the pending subsequences. For the first stage, the subsequence is only the full sequence. So $il = 0$, $ir = N - 1$ and the top pointer of the stack $jstack = 0$. For a subsequence, when the length is shorter than some size *LQS*, it is faster to use the straight insert sort. To avoid the “worst case” running time in the quick sort, which usually happens when the input sequence is already in order, the median of the first, middle, and last elements of the current sub-sequence is used as the “partitioning element”.

The partitioning process is carried out in the “do-while” loops for indices i and j . For the selected partitioning element a , we first scan the i pointer up until we find an element $> a$. Then we scan another j pointer down until we find an element $< a$. Then these two elements are clearly out of place for the final partitioned array, so we exchange them. This process is continued until the pointers cross. Since we do not care about the order of the sub-sequences larger than M , we only need to push the “left” and “right” pointers lower than M to the stack. These pointers are popped out from the stack only when the sub-sequence is short enough for the final “insert-sort” procedure. This not only reduces the size of the stack, but also reduces the latency in processing the subsequences higher than M . To generate the order indices, the elements in the order vector $I[N]$ exchange whenever the sequence elements exchange. Because $M \ll MC$ in general, the complexity reduction is significant. Simulation shows that the complexity is reduced dramatically to the order of the $\mathcal{O}(MC + \text{const})$. The number of cycles is slightly higher than the sequence length with design overheads in branching.

D. Embedded Insert Sort

In this section, we propose an alternative architecture based on the stochastic feature of the metric sequence. In this architecture, we embed the insert sort in the metric computation procedure. When a new metric is computed, the new metric is compared with the existing sequence to see if it is smaller than the current sequence. The existing sequence only keeps the current M smallest metrics, which requires a buffer of length M . The new metric starts the comparison from the top of the buffer. If the buffer is still not full, the new metric will find a place to insert into the buffer. If it is bigger than the top, then just add it to the top. Otherwise, compare it with the other entries from the top until it finds a smaller entry. Then it will insert into the found place. On the other hand, if the buffer is already full, just drop the biggest metric after the comparison and insertion. Note that in this case, if the new metric is the biggest one, it only needs to be compared with the top entry in the buffer and be dropped.

For the first entry, there is no comparison but an writing operation to the buffer. For the newly computed metric, we need to first decide if the buffer is full or not. This is based on the edge detection to find whether $i < M$. When the buffer is not full, the top pointer of the buffer is the current index of the computed metric. Otherwise, when the buffer is full, the top pointer is the $M - 1$. Then to do the insert sort, we need to first test if the new entry is smaller than the top entry, i.e., the current max value of the existing buffer entries. Only when the new value is smaller than the top entry do we need to find the place to insert the new value and make the bigger entries float in the buffer. This comparison will begin from the second largest entry in the buffer. Here we need to test two cases. If the buffer is not full, the top entry needs to be kept. Otherwise, it is dropped automatically by overwriting.

Fig. 3. Metric increasing trend of the branch metric in stage 3.

E. Metric update Trend

The benefit of this architecture is that we only need to keep a M -size buffer instead of $M \times C$ memory. However, in the M -algorithm, the latency of this sorting architecture depends on the stochastic feature of the new metric sequence. In the best case, if the sequence is already ordered, the new metric only need to be put at the top (buffer is not full) or dropped (after buffer is full). In the worst case, if the sequence is ordered reversely, each new metric needs to be compared with all the entries in the buffer to find its place at the bottom of the buffer.

We found that if the survivor branches from the previous antenna are already sorted, the branch with smaller survivor metric tends to generate C smaller derived metrics at the current antenna stage. So after more intensive comparison and sorting for the new metrics of the early survivor branches, the new metrics of the later survivor branches tends to be bigger than most or even all the entries in the buffer and tends to be dropped after comparing with only the top of the buffer. This can save the memory access dramatically with comparison on the fly. The metric update trend of the sequence is verified in the metric increasing trend in Fig. 3 for $M = 64$ and 64-QAM, where the moving average is increasing.

IV. SIMULATION AND EMULATION

A. Fixed-point Emulation Performance

To overcome the simulation time bottleneck, a FPGA based hardware accelerator [7] is proposed to shorten the simulation time with the bottleneck part of the receiver implemented in FPGA hardware. To support a wide range of the simulation specification, we choose the input word length to be 10 bits. The simulation results for 4×4 configuration are shown in Fig. 4. For the 4×4 scenario, $M = 4$ gives promising performance gain over the $M = 1$ case for QPSK. $M = 16$ offers very small performance gain over $M = 8$ for the 16-QAM. The measured FPGA BER curves overlap with the floating-point curves in all cases, which verifies that the fixed-point FPGA implementation is functionally correct.

Fig. 4. Measured FPGA Bit Error Rate performance: 4×4 , QPSK.

Fig. 5. Measured run-time profile original C-MEX: 4×4 , 64-QAM.

Fig. 6. Measured FPGA run-time profile: 4×4 , 64-QAM.

Fig. 7. Measured run-time comparison: 4×4 , 64-QAM.

B. Run-time Acceleration

In this section, we analyze the run-time profile in a simplified 4G MIMO-OFDM simulation chain in matlab. The top five most time consuming functions in the simulation chain are shown in Fig. 5 and 6 for both the original C-MEX design and the FPGA implementation for 64-QAM and 2×2 configuration. The run-time is obtained by the matlab “profile” function. Function “fhardqrdm_” is the receiver function including all “m_mex_orig”, “channel”, “qr” and “mapping” sub-functions, where the QR-decomposition calls the matlab built-in function. It is shown that for the original floating-point C-MEX implementation, the C-MEX implementation of the M-search function “m_mex_orig” consumes most of the simulation time. Moreover, all the other functions consumes negligible time compared with the M-search function. So only the M-search function is implemented in FPGA hardware with the proposed complexity optimizations. In this case, 5 parallel processing elements are running in the Virtex-II V3000 at 33 MHz clock, which is competing with the 1.5 GHz clock of the PC processor. It is shown that the “mloopfpga_mex” now consumes a much smaller portion of the simulation time, which does not increase dramatically with higher M . The speedup is up to $100\times$ for the emulated case.

V. CONCLUSION

In this paper, we present a novel reduced complexity QRD-M for MIMO-OFDM with partial quick sort and embedded insert sort architectures. The complexity is dramatically reduced both for the metric update and tree search process.

The related VLSI architectures are evaluated and implemented in a compact form factor FPGA platform to accelerate the simulation time significantly.

REFERENCES

- [1] G. J. Foschini, Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas, Bell Labs Tech. J., pp. 41-59, 1996.
- [2] 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, in 1998 URSI International Symposium on Signal, Systems, and Electronics, pp. 295-300, 1998.
- [3] Y. Li, J. H. Winters and N. R. Sollenberger, MIMO-OFDM for wireless communications: signal detection with enhanced channel estimation, IEEE Transactions on Communications, vol. 50, pp. 1471-1477, Sept. 2002.
- [4] J. Yue, K. J. Kim, J. D. Gibson and R. A. Iltis, Channel estimation and data detection for MIMO-OFDM systems, IEEE Global Telecommunications Conference, vol. 22, no. 1, pp. 581-585, Dec 2003.
- [5] T. M. Aulin, Breadth-first maximum likelihood sequence detection: Basics, IEEE Transactions on Communications, vol. 47, pp. 2082-216, Feb. 1999.
- [6] G. H. Golub and C. F. V. Loan, Matrix Computations. The Jones Hopkins University Press, 1996.
- [7] Y. Guo and D. McCain, Compact FPGA Hardware Accelerator for Functional Verification and Rapid Prototyping of 4G Wireless Systems, Proceeding of IEEE Asilomar Conference, Monterey, CA, Nov. 2004.
- [8] L. Wei, L. Rasmussen and R. Wyrwas, Near optimum tree-search detection schemes for bit-synchronous multiuser CDMA systems over Gaussian and two-path Rayleigh fading channels, IEEE Trans. Commun., vol. 45, pp. 691-700, June 1997.
- [9] Y. Guo, G. Xu, D. McCain, J. R. Cavallaro, Rapid scheduling of efficient VLSI architectures for next-generation HSDPA wire-less system using Precision-C synthesizer, Proc. IEEE Intl. Workshop on Rapid System Prototyping'03, San Diego, CA, pp. 179-185, June 2003.