Flexible N-Way MIMO Detector on GPU

Michael Wu, Bei Yin, Joseph R. Cavallaro
Electrical and Computer Engineering
Rice University, Houston, Texas 77005
{mbw2, by2, cavallar}@rice.edu

***Abstract*—This paper proposes a flexible Multiple-Input Multiple-Output (MIMO) detector on graphics processing units (GPU). MIMO detection is a key technology in broadband wireless system such as LTE, WiMAX, and 802.11n. Existing detectors either use costly sorting for better performance or sacrifice sorting for high throughput. To achieve good performance with high thoughput, our detector runs multiple search passes in parallel, where each search pass detects the transmit stream with a different permuted detection order. We show that this flexible detector, including QR decomposition preprocessing, outperforms existing GPU MIMO detectors while maintaining good bit error rate (BER) performance. In addition, this detector can achieve different tradeoff between throughput and accuracy by changing the number of parallel search passes.**

I. Introduction

Multiple-Input Multiple-Output (MIMO) is a key technique in many high throughput wireless standards such as 3GPP LTE, WiMAX and 802.11n. However, as the received signals contain a mixture of the transmitted signals in the air, the destination needs to perform MIMO detection to recover the original transmitted signals.

The optimal MIMO detection method is maximum likelihood (ML) detection, which is an integer least-squares problem that can be solved with an exhaustive search. However, the complexity of an exhaustive search is exponential. As a result, ML detection is not suitable for practical implementations as the destination has strict area and timing requirements. Subsequently, several suboptimal algorithms have been proposed. These algorithms have significantly reduced complexity and can be viewed as different tree search strategies. The detection algorithms can be divided into two categories: depth-first algorithms such as depth-first sphere detection [1], and breadth-first algorithms such as K-best [2]. The main issue of depth-first sphere detection is that the number of tree nodes visited vary with signal to noise ratio (SNR). The algorithm visits large number of nodes in the low SNR, while visits a small number of nodes in the high SNR. As a result, the throughput of this detection algorithm varies with SNR, which is an undesirable feature for real systems. An attractive alternative is K-best detection. This algorithm has a fixed throughput, because it searches a fixed number of tree nodes independent of SNR. However, to achieve a comparable performance to exhaustive search, a large K value is required.

In this paper, we aim to leverage massively parallel computational power in off-the-shelf GPUs to achieve high throughput MIMO detection. Moreover, compared to ASIC and FPGA, an software implementation is attractive since it can support a wide range of parameters such as modulation order and MIMO configuration. The main challenge of a K-best design is the global sort at each step of the algorithm. This is undesirable on GPU as sorting require synchronization and large amount of memory for large K. To reduce the sorting complexity, a number of modified sort-free algorithms have been developed. In SSFE [3], instead of sorting N values to find the best K values, the workload is first partitioned into M arrays, where M is the modulation order. Fast enumeration is used to find the best K/M values for each sub-array without sorting each sub-array. However, eliminating the global sort reduces the accuracy of the detector. To recover the performance loss, we use parallel search passes, where each search uses a different permuted antenna detection order [4, 5]. Since each search pass performs the same set of operations on permuted data, this algorithm remains highly data parallel and well suited for GPU.

Our contributions are the following. We show that the proposed design achieves different tradeoffs between throughput and accuracy by modifying the number of parallel tree searches. We show this design achieves higher throughput than other GPU implementations [6, 7] while maintaining equal or better accuracy. In addition, QR decomposition is a required step which is omitted in other papers. In this paper, we complete the design by implementing modified Gram-Schmidt Orthogonalization to perform QR decomposition. We note that our design is different from other decomposition methods such as [8]. Where as the existing designs focus on a single large matrix, our implementation performs QR decomposition on many small dense matrices in parallel.

This paper is organized as follows: Section 2 gives a quick overview of CUDA. Section 3 gives an overview of the system model. Section 3 describes the proposed detection algorithm on GPU. Section 4 presents the performance of the detector. Finally, we conclude in section 5.

II. Overview of CUDA

Computer Unified Device Architecture (CUDA) [9] adopted in this work is widely used to program massive parallel computing applications. In Nvidia Fermi architecture, GPU consists of multiple stream multiprocessors (SM). Each SM consists of 32 pipelined cores and two instruction dispatch units. During execution, each dispatch unit can issue a 32 wide single instruction multiple data (SIMD) instruction which is executed on a group of 16 cores. CUDA device has a large amount (> 1GB) of off-chip device memory (or global memory). As latency to off die device memory is high, fast on-chip resources, such as registers, shared memory and constant memory can be used in place of off-chip global memory to keep the computation throughput high.

In this model, if a task is executed several times, independently, over different data, the task can be mapped into a kernel, downloaded to a GPU and executed in parallel on many different threads. The programmer defines this kernel function, a set of common operations. At runtime, the kernel spawns a large number of threads blocks, where each thread block contains multiple threads. The execution of a kernel on a GPU is distributed according to a grid of thread blocks with adjustable dimensions. Each thread can select a set of data using its own unique ID and executes the kernel function on the set of data. Threads execute independently in this model. However, threads within a block can synchronize through a barrier and writing to shared memory. In contrast, thread blocks are completely independent and can be synchronized by terminating the kernel and writing to global memory.

During kernel execution, multiple thread blocks can be assigned to a SM and are executed concurrently. CUDA divides threads within a thread block into blocks of 32 threads. These 32 threads are executed in lockstep using the same common instruction, a WARP instruction. Each instruction dispatch unit on a SM can issue a WARP instruction whose operands are ready. A stall can occur for device memory reads and instruction dependencies. To mask the stall, the instruction dispatch unit can switch and issue an independent WARP instruction from the same thread block or another concurrent thread block with zero-overhead. In addition, stalls can be minimized by using fast on-die memory resources.

Registers, shared memory, and constant memory can reduce memory access time by reducing global memory access. Registers and shared memory are on-chip resources. Shared memory is slower than registers, but can be accessed by threads within a thread block. However, shared memory on each SM is banked 32 ways. It takes one load or store if all threads access the same bank (broadcast) or none of the threads accesses the same bank. Random layout with some broadcast and some one-to-one accesses will be serialized.

III. MIMO System Model

For an $N_t \times N_t$ MIMO system, the source transmits N_t signals and the destination receives signals on N_t antennas. The received signal, $\mathbf{y} = [y_0, y_1, ..., y_{N_t-1}]^T$, is modeled by:

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

where $\mathbf{H} = [\mathbf{h_0}, \mathbf{h_1}, ..., \mathbf{h_{N_t-1}}]$ is the $N_t \times N_t$ channel matrix. Assume a flat fading Rayleigh fading channel, where each element of $\mathbf{H}$, h_{ij}, is an i.d.d. zero mean circular symmetric complex Gaussian (ZMCSCG) random variable with σ_h^2 variance. The vector $\mathbf{n} = [n_0, n_1, ..., n_{N-1}]$ is the additive noise, where each element n_j is a ZMCSCG random variable with $\sigma_n^2/2$ variance per dimension. Each element of $\mathbf{s}$, s_i, is an complex element drawn from a finite alphabet $\mathbf{\Omega}$ with cardinality M and average power E_s per symbol. For example, the constellation alphabet for QPSK is $\{-1-j, -1+j, 1-j, 1+j\}$ with $M = 4$. Given a binary vector $\mathbf{x} = [x_0, x_1, x_2...x_{L-1}]^T$, where $L = \log_2 M \cdot N_t$, the function $\mathrm{map}(\cdot)$ translates the binary vector $\mathbf{x}$ onto $\mathbf{s} = [s_0, s_1, ..., s_{N_t-1}]^T$.

We can obtain an equivalent system model in the real domain by performing real-valued decomposition:

$$\begin{pmatrix} \Re(\mathbf{y}) \\ \Im(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} \Re(\mathbf{H}) & -\Im(\mathbf{H}) \\ \Im(\mathbf{H}) & \Re(\mathbf{H}) \end{pmatrix} \begin{pmatrix} \Re(\mathbf{s}) \\ \Im(\mathbf{s}) \end{pmatrix} + \tilde{\mathbf{n}}, \tag{2}$$

We then obtain an equivalent system model in the real domain through modified real value decomposition (MRVD). We permutate the channel matrix such that the in-phase and the quadrature parts of the same complex symbol are adjacent neighbors [10]:

$$\begin{pmatrix} \Re(y_0) \\ \Im(y_0) \\ \vdots \\ \Re(y_{N_t-1}) \\ \Im(y_{N_t-1}) \end{pmatrix} = \tilde{\mathbf{H}} \begin{pmatrix} \Re(s_0) \\ \Im(s_0) \\ \vdots \\ \Re(s_{N_t-1}) \\ \Im(s_{N_t-1}) \end{pmatrix} + \begin{pmatrix} \Re(n_0) \\ \Im(n_0) \\ \vdots \\ \Re(n_{N_t-1}) \\ \Im(n_{N_t-1}) \end{pmatrix} \tag{3}$$

$$\tilde{\mathbf{y}} = \tilde{\mathbf{H}}\hat{\mathbf{s}} + \tilde{\mathbf{n}} \tag{4}$$

MRVD doubles the number of elements in each vector and doubles both dimensions of $\tilde{\mathbf{H}}$. Furthermore, each element of the equivalent transmit vector, $\tilde{s}_i$, is an element drawn from a smaller finite alphabet $\mathbf{\Omega}'$ with cardinality $Q = \sqrt{M}$. For example, the real value decomposed constellation alphabet for QPSK is $\{-1, 1\}$ and $Q = 2$.

Given $\tilde{\mathbf{y}}$ and the channel matrix $\tilde{\mathbf{H}}$, the goal of the soft-output MIMO detector at a MIMO receiver is to compute the logarithmic a-posteriori probability (APP) ratio, $L_D(x_k|\tilde{\mathbf{y}}, \tilde{\mathbf{H}})$, per bit. Assuming no prior knowledge of the transmitted bits, the soft-output value per bit can be approximated with the following equation using max-Log approximation [11].

$$L_D(x_k|\tilde{\mathbf{y}}, \tilde{\mathbf{H}}) = \min_{\mathbf{x} \in \mathbb{X}_{k,-1}} \frac{\left\| \tilde{\mathbf{y}} - \tilde{\mathbf{H}}\tilde{\mathbf{s}} \right\|_2^2}{2\sigma_n^2} - \min_{\mathbf{x} \in \mathbb{X}_{k,+1}} \frac{\left\| \tilde{\mathbf{y}} - \tilde{\mathbf{H}}\tilde{\mathbf{s}} \right\|_2^2}{2\sigma_n^2}, \tag{5}$$

where $\mathbb{X}_{k,-1}$ is the list of all binary vectors with the k^{th} component equal to -1, $\mathbb{X}_{k,+1}$ is the list of all binary vectors with the k^{th} component equal to $+1$, and $\tilde{\mathbf{s}} = \mathrm{map}(\mathbf{x})$.

Instead of searching through the set of all possible binary vectors to compute $L_D(x_k|\tilde{\mathbf{y}}, \tilde{\mathbf{H}})$, a soft-output MIMO detector finds a smaller set of transmit vectors, or a candidate list, $\mathcal{L}$, by excluding unlikely vectors. To compute $L_D(x_k|\mathbf{y}, \mathbf{H})$, the candidate list is divided into $\mathcal{L}_{k,-1}$ and $\mathcal{L}_{k,+1}$, where $\mathcal{L}_{k,-1}$ is the list of candidates with the k^{th} bit equal to -1 and $\mathcal{L}_{k,+1}$ is the list of candidates with the k^{th} bit equal to $+1$. The list $\mathcal{L}_{k,-1}$ is used to generate the hypothesis, while the list $\mathcal{L}_{k,+1}$ is used to generate the counter-hypothesis.

$$L_D(x_k|\tilde{\mathbf{y}}, \tilde{\mathbf{H}}) \approx \underbrace{\min_{\mathbf{x} \in \mathcal{L}_{k,-1}} \frac{\left\| \tilde{\mathbf{y}} - \tilde{\mathbf{H}}\tilde{\mathbf{s}} \right\|_2^2}{2\sigma_n^2}}_{hypothesis} - \underbrace{\min_{\mathbf{x} \in \mathcal{L}_{k,+1}} \frac{\left\| \tilde{\mathbf{y}} - \tilde{\mathbf{H}}\tilde{\mathbf{s}} \right\|_2^2}{2\sigma_n^2}}_{counter-hypothesis}. \tag{6}$$

IV. Soft-output N-Way MIMO Detector on GPU

In this section, we will explain the algorithm as well as the corresponding implementation on GPU for one MIMO detection problem. The implementation consists of two kernels. One kernel performs the QR decomposition. The other kernel first

Algorithm 1 Modified Gram-Schmidt for kth thread

```
1) Input: $\mathbf{y}, \mathbf{H}$
2) Initialization:
    a) $s = 0$ //$s$ is in shared memory
    b) Fetch $\mathbf{y}$ and $\mathbf{H}$ to construct $\mathbf{V} = [\tilde{\mathbf{H}}|\tilde{\mathbf{y}}]$ in shared memory
3) for step i = 0 to 2N-1 do
4)     if (k = i)
5)         $\mathbf{E}_{i,i} = \mathbf{v}_i^* \mathbf{v}_i$
6)         $s = 1/\sqrt{\mathbf{E}_{i,i}}$
7)     end if
8)     $\_\_syncthreads()$
9)     $\mathbf{V}_{k,i} = \mathbf{V}_{k,i} \cdot s$
10)    $\_\_syncthreads()$
11)    if (k >= i)
12)        $\mathbf{E}_{i,k+1} = \mathbf{v}_i^* \mathbf{v}_{k+1}$
13)        $\mathbf{v}_{k+1} = \mathbf{v}_{k+1} - \mathbf{v}_i \cdot \mathbf{E}_{i,k+1}$
14)    end if
15) end for
```

performs the candidate list search. The kernel then uses the list to generate a hypothesis and a counter-hypothesis for each transmitted bit which are used to compute a soft-output value for each transmitted bit.

Although the description is for one MIMO detection problem, a typical wireless system divides the available bandwidth into many orthogonal independent subcarriers, where each is an independent MIMO detection problem. Our implementation performs MIMO detection on many subcarriers in parallel using hundreds of independent thread-blocks to achieve high performance.

A. QR decomposition

Given $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{H}}$, we first perform QR decomposition on $\tilde{\mathbf{H}}$ to obtain an equivalent system model, where the squared Euclidean distance of a transmit vector $\tilde{\mathbf{s}}$, is:

$$\left\|\tilde{\mathbf{y}} - \tilde{\mathbf{H}}\tilde{\mathbf{s}}\right\|_2^2 = \|\hat{\mathbf{y}} - \mathbf{R}\tilde{\mathbf{s}}\|_2^2 . \qquad (7)$$

where $\mathbf{R}$, an upper-triangular matrix and $\hat{\mathbf{y}} = \mathbf{Q}^T\tilde{\mathbf{y}}$ is the effective received vector.

We implemented Modified Gram-Schmidt orthogonalization to perform QR decomposition. We spawn $2N_t$ threads to perform one QR decomposition. The steps of the kernel are summarized in Algorithm 1. At the start of the kernel, the $2N_t$ threads fetch the complex inputs, $\mathbf{y}$ and $\mathbf{H}$, from device memory and construct a real-value extended matrix $\mathbf{V} = [\tilde{\mathbf{H}}|\tilde{\mathbf{y}}] = [\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_N]$ in shared memory. We perform QR decomposition on $\mathbf{V}$ which results in an extended matrix $\mathbf{E} = [\mathbf{R}|\hat{\mathbf{y}}]$ stored in device memory. The QR decomposition consists of $2N_t$ Gram-Schmidt iterations. The ith iteration induces zeros below the ith element on the diagonal of $\mathbf{V}$ and constructs the ith row of $\mathbf{E}$. Each iteration consists of a serial and a parallel section. Line 4-7 is the serial section, in which the ith thread first constructs $\mathbf{E}_{i,i}$ by computing the squared Euclidean distance of $\mathbf{v}_i$ and computes the corresponding

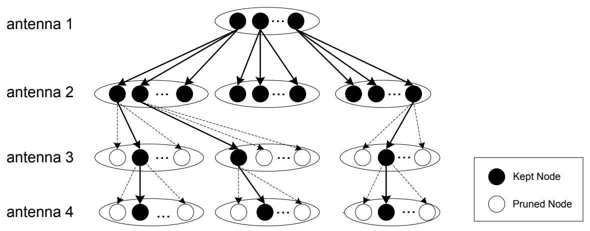


Figure 1. An example of the search process for a 2x2 16-QAM MIMO system

scaling factor s. Subsequent steps of the iteration are computed in parallel. Line 9 first computes the ith orthogonal projection using all $2N_t$ threads in parallel. Lines 11-15 assign one thread per column to the $2N_t - i + 1$ columns on the right of the ith column. Line 12 first constructs the remaining elements in the ith row of $\mathbf{E}$ in parallel. In Line 13, thread k updates $\mathbf{v}_k$ by subtracting the projection of $\mathbf{v}_k$ on to the $\mathbf{v}_j$ from $\mathbf{v}_k$. After each iteration, the variable i increases by one, effectively decreasing the row dimension and the column dimension of the $\mathbf{V}$ by one. When the number of rows remaining reaches 0, we have obtained $\mathbf{R}$ and $\hat{\mathbf{y}}$, which is stored the matrix $\mathbf{E}$.

Since the dimensions of the extended matrix $\mathbf{V}$ are small for typical MIMO systems, the matrix can be stored in shared memory for efficient retrieval, reducing the number of slower device memory accesses. Furthermore, the memory accesses are very regular for this kernel and can be served effectively by the shared memory. The matrix $\mathbf{V}$ has a row-major layout. Since the row dimension of $\mathbf{V}$ is an odd number, $2N_t + 1$, and shared memory is banked 32 ways, rows of $\mathbf{V}$ are in different banks. As a result, shared memory accesses by multiple threads in line 9 do not result in memory conflicts. In lines 12-13, threads access $\mathbf{v}_i$ and a column of $\mathbf{V}$ in parallel. Both memory accesses do not result in memory bank conflicts. Parallel memory accesses to $\mathbf{v_i}$ are handled effectively by shared memory read broadcast. Since adjacent columns are stored in different memory banks, parallel access to a column of $\mathbf{V}$ also does not result in memory bank conflict.

B. 1-Way MIMO Detection

The MIMO Detector consists of two steps. First we search for the likely candidate vector with small squared Euclidean distance. Second, we use the candidates to compute the hypothesis and the counter-hypothesis per transmitted bit. The differences between the hypotheses and the counter-hypotheses are the APP ratio per bit.

1) Candidate Search: This search algorithm searches for candidate vectors with small Euclidean distances in a greedy fashion to generate a small candidate list. Since $\mathbf{R}$ is upper triangular, the search algorithm evaluates the transmit vector level by level backwards from level $N_t - 1$. The search algorithm can be viewed as a tree traversal where the branches of the tree are pruned level by level until there are a few complete paths left. Figure 1 is a complete search tree for a 2×2 16-QAM MIMO system. In this search algorithm, all branches in the first two levels are kept. As a result, the first two levels of the tree are fully expanded. For the subsequent tree levels, the search algorithm only keeps the best outgoing

Algorithm 2 The kth thread search for kth candidate

```
1) Input: $\mathbf{E} = [\mathbf{R}|\hat{\mathbf{y}}]$
2) Initialization:
      a) $d = 0$, $Q = \sqrt{M}$
      b) $\mathbf{p}^k = [0, 0, ..., 0, \Im(\Omega_k), \Re(\Omega_k)]$
3) $d = d + (\hat{\mathbf{y}}_{N_t-1} - \mathbf{R}_{N_t-1,N_t-1}\mathbf{p}^k_{N_t-1})^2$,
4) $d = d + (\hat{\mathbf{y}}_{N_t-2} - \mathbf{R}_{N_t-2,N_t-1}\mathbf{p}^k_{N_t-1} - \mathbf{R}_{N_t-2,N_t-2}\mathbf{p}^k_{N_t-2})^2$
5) for step i = $N_t - 3$ to 0 do
6)     $b_i = \hat{\mathbf{y}}_i$,
7)     for step j = $N_t - 1$ to $i + 1$
8)         $b_i = b_i - \mathbf{R}_{i,j} \cdot \mathbf{p}^k_j$
9)     end for
       // Find the best outgoing node
10)    $\gamma = {}^{b_i}/_{\mathbf{R}_{i,i}}$
11)    $\mathbf{p}^k_i = round\left(\frac{1}{2}(\gamma + Q - 1)\right) \cdot 2 - Q + 1$
12)    if $(|\mathbf{p}^k_i| > Q - 1)$     $\mathbf{p}^k_i = sign(\mathbf{p}^k_i) \cdot (Q - 1)$
       // Update squared Euclidean distance
13)    $d = d + (b_i - \mathbf{R}_{i,i} \cdot \mathbf{p}^k_i)^2$
14) end for
```

path. The candidate list consists of the surviving paths at the last level.

Given $\mathbf{p}^j$, the set of nodes along the path from the root node to the jth node on level t, $w^{<t-1>}_{j,k}$, the partial squared Euclidean distance from jth node on level t to the kth node on level $t-1$ can be computed as,

$$w^{<t-1>}_{j,k} = ||\hat{\mathbf{y}}_{t-1} - \sum_{i=N_t-1}^{t} \mathbf{R}_{k,i}\mathbf{p}^j_i - \mathbf{R}_{t-1,t-1}s_k||^2_2, \quad (8)$$

$$= ||b_{t-1}(\mathbf{p}^j_i) - \mathbf{R}_{t-1,t-1}s_k||^2_2. \quad (9)$$

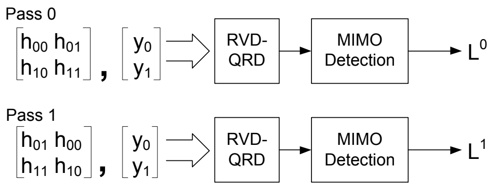
The best connected node in level $t-1$ that minimizes $w^{<t-1>}_{j,k}$ is simply the closest constellation point in Ω' to $\gamma = {}^{b_{t-1}(\mathbf{p}^j_i)}/_{\mathbf{R}_{t-1,t-1}}$. Suppose node k is the best node, the total squared Euclidean distance after the best node is found at antenna level $t-1$ is:

$$d = d + w^{<t-1>}_{j,k}. \quad (10)$$

The steps of the kernel are summarized in Algorithm 2. We spawn M threads, one thread per modulation point, to perform the tree search. We assign the kth modulation point in our finite alphabet, Ω_k, to thread k and we initialize the distance, d, to 0. Using the initial modulation point, each thread first updates the distance by computing the squared Euclidean distance using Ω_k as shown in Lines 3-4. At the end of the update, the first two levels of the tree are fully expanded. In the subsequent levels, the search algorithm only keeps the best outgoing path. Lines 6-9 compute the partial squared Euclidean distance. Line 10 uses the partial Euclidean distance to compute γ. The best node can be implemented with a simple round function followed by a threshold function on γ which is shown on lines 11-12. Line 13 updates the squared Euclidean distance. This process continues until we have reached the last tree level.

Algorithm 3 The kth thread updates kth hypothesis and counter-hypothesis

```
1) $\mathbf{b}^k$ = demod($\mathbf{p}^k$), $\mathbf{d}_k = d$
2) $\_\_syncthreads()$
3) if $(k < N_t \cdot \log(M))$
4)     Initialization: $h_k = 9999$,   $c_k = 9999$
5)     for step j = 0 to M-1 do
6)         if ($k$th bit of $\mathbf{b}^j$= 1) and ($\mathbf{d}_k$<$h_k$)
7)             $h_k = \mathbf{d}_k$
8)         else if ($k$th bit of $\mathbf{b}^j$= -1) and ($\mathbf{d}_k$<$c_k$)
9)             $c_k = \mathbf{d}_k$
10)        end if
11)    end for
12) end if
```



Pass 0
$\begin{bmatrix} h_{00} & h_{01} \\ h_{10} & h_{11} \end{bmatrix}$, $\begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$
RVD-QRD
MIMO Detection
L^0
Pass 1
$\begin{bmatrix} h_{01} & h_{00} \\ h_{11} & h_{10} \end{bmatrix}$, $\begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$
RVD-QRD
MIMO Detection
L^1

Figure 2. Proposed Detector for a 2x2 MIMO System.

2) Hypothesis and Counter-hypothesis Generation: With the candidate list, the detector can generate the hypothesis and counter-hypothesis for each transmitted bit. Algorithm 3 summarizes the work required to generate the hypothesis and counter-hypothesis per bit. Each thread demodulates $\mathbf{p}^k$ into a binary vector $\mathbf{b}^k$ which is stored in shared memory. In addition, the kth thread writes the Euclidean distance for the kth path into $\mathbf{d}_k$ stored in shared memory. Both steps are shown on line 1. We use $N_t \cdot \log(M)$ threads to generate the hypothesis and the counter-hypothesis per bit. We assign one thread per bit, the kth thread looks at the kth bit. The threads look at the binary vector $\mathbf{b}^j$ one by one. If the kth bit is equal to 1 and the $\mathbf{d}_k$ is the smaller than the current hypothesis, the kth hypothesis is updated. Likewise, if the kth bit is equal to 0 and the $\mathbf{d}_k$ is smaller than the current counter-hypothesis, the kth counter-hypothesis is updated.

To improve performance, we unroll the nested loops which reduce the total number of instructions. To improve memory access time, we take advantage of the fact that there are no data dependencies between the threads. Instead of using shared memory, we store the path history $\mathbf{P}$ for each thread directly in registers. We found storing the array in registers to be faster since operations with shared memory as an operand is known to be slower [12]. Using registers instead of shared memory also eliminates memory address computation.

C. N-Way Parallel MIMO Detection

A MIMO detector that employs QR decomposition followed by a single tree search is an SSFE detector. We find an SSFE detector performs up to $2dB$ worse compared to ML detection shown in section V. To improve performance, we construct a larger candidate list by performing multiple tree searches in parallel. An instance of the proposed algorithm, two search passes for a 2x2 MIMO 16-QAM system, is shown in Fig. 2. The inputs for each pass are the same, consisting of the channel matrix $\mathbf{H}$ and the received vector $\mathbf{y}$. Since the detection order affects the performance of the detector and the optimal antenna detection order is not known, each pass uses a different antenna detection order to generate different candidates lists. A different antenna detection order can be obtained by a simple circular rotation of columns of $\mathbf{H}$. In this design, we can run up to N parallel passes to generate N parallel M length candidate lists.

In our implementation, for an input pair $\mathbf{H}$ and $\mathbf{y}$, we spawn $2N_T \cdot N$ threads to perform the N QR decompositions in parallel. For the N-way parallel search, we spawn $2M \cdot N$ threads to perform the N parallel search passes. Note that the QR decompositions and the parallel searches for an input pair are completely independent. Threads corresponding to an input block reside in the same thread-block. In the case where the number of threads per thread-block is not an integer of 32 (the dimension of a WARP instruction). We pack multiple problems into the same thread-block to improve efficiency. This is especially the case when $N = 1$. For example, for a 4×4 MIMO 16-QAM system, the number of threads required for QR decomposition is 8 threads and the number of threads required for MIMO detection is 16 threads. We pack at least 4 QR decomposition problems into one thread block and at least two 16-QAM detectors into one thread block to ensure WARPs for the thread-block are fully occupied.

D. LLR Computation

After the N-way parallel search, each search generates the hypothesis and the counter-hypothesis for each transmitted bit. This results in N hypotheses and counter-hypotheses per bit. We then merge N hypotheses per bit into one hypothesis and N counter-hypotheses into one counter-hypothesis by assigning one thread to the hypothesis and one thread to the counter-hypothesis per bit. According to equation 6, since the hypothesis and counter-hypothesis are the global minimums, each thread searches across N hypotheses or N counter-hypothesis to find the minimums. The difference between the final hypothesis and the final counter-hypothesis is the soft-output value for the bit.

V. Performance

In this section, we first compare the frame error rate (FER) performance of our detector implementation against other detectors. We then look at the throughput performance of the detector and show that it is faster than other detectors.

A. FER Performance

We compared the FER performance of the N-way parallel MIMO detector against the soft-output Trellis detector [7], the K-best detector as well as ML detection which is exhaustive search. In our simulation, we first generate a random input

Table I
QR Decomposition kernel time for 8400 MIMO symbols

	N=1	N=2	N=3	N=4
2×2	0.030ms	0.052 ms	-	-
4×4	0.137ms	0.257 ms	0.399 ms	0.501 ms

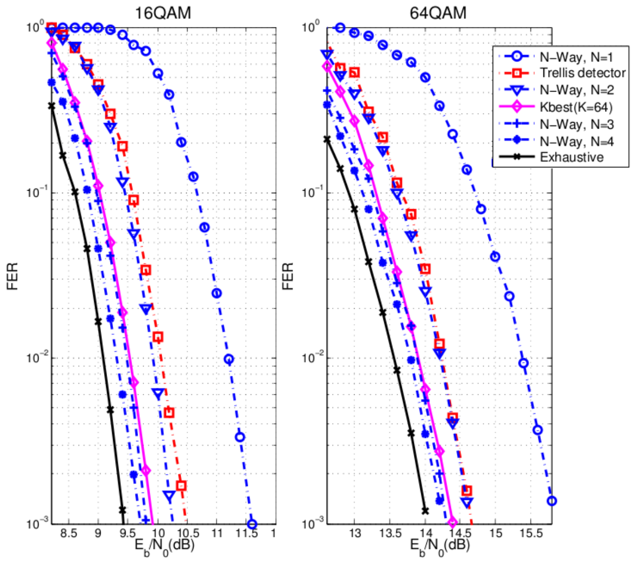


Figure 3. 4x4 FER Performance

binary vector. After modulating the binary vector into a MIMO symbol, the symbol is passing through a random flat fading Rayleigh channel. The detector performs QR decomposition followed by soft-output detection to generate one soft-output value per transmitted bit. The soft output of the detector is then fed to a length 2304, rate 1/2 WiMAX LDPC decoder, which performs up to 20 decoding iterations. For the K-best detector, we chose a large K value of 64 for 16-QAM and 256 for 64-QAM. For the N-way parallel MIMO detector, we test different instances of the detector from $N = 1$ to $N = 4$. The detectors use an LLR clipping value of 8 for all the detector configurations with the exception of $N = 4$ where LLR clipping was not required.

Figure 3 compares the performance of detectors for 16-QAM and 64-QAM. The trends are similar for both plots. The N-way parallel MIMO detector is equivalent to SSFE when $N = 1$. In both cases, $N = 1$ performs poorly compared to the other detectors. As we increase N, we improve performance of the detector since a larger list increases the probability of finding the best hypothesis and the counter-hypothesis per transmitted bit. For $N = 2$, we outperform the soft-output Trellis detector in both cases. For N = 3, we perform similar to the large K-best detector. For N = 4, the N-way scheduled MIMO detector's performance is close to exhaustive search. The results suggest modifying N is a good way to scale the detector performance.

B. Throughput Performance

To measure the throughput performance of our implementation, we used an Nvidia GeForce GTX 560 Ti graphics card with 448 shaders running at 1464MHz and 1280MB of DDR5 running at 1900MHz. We used 8400 MIMO symbols, the number of MIMO symbols in a slot for 20MHz LTE channel. In our experiment, the execution time of our runs is averaged over 1000 runs. We first look at the runtime of QR decomposition and the runtime of N-Way detection in isolation without considering transport time. We then look at the runtime of the entire design considering transport time

Table II
MIMO DETECTION KERNEL TIME FOR 8400 MIMO SYMBOLS

	N=1	N=2	N=3	N=4
2×2 16-QAM	0.0768ms/834.1Mbps	0.159ms/402.2Mbps	-	-
4×4 16-QAM	0.164ms/782.5Mbps	0.332ms/386.14Mbps	0.500ms/256.2Mbps	0.665ms/192.5Mbps
2×2 64-QAM	0.524ms/183.5Mbps	1.040ms/92.4Mbps	-	-
4×4 64-QAM	0.833ms/230.7Mbps	1.658ms/115.9Mbps	2.474ms/77.7Mbps	3.297ms/58.3Mbps

Table III
TOTAL RUNTIME FOR 8400 MIMO SYMBOLS

	N=1	N=2	N=3	N=4
2×2 16-QAM	0.342ms/195.6Mbps	0.462ms/145.0Mbps	-	-
4×4 16-QAM	0.739ms/181.3Mbps	1.048ms/127.8Mbps	1.349ms/99.3Mbps	1.629ms/82.3Mbps
2×2 64-QAM	0.818ms/122.3Mbps	1.363ms/73.3Mbps	-	-
4×4 64-QAM	1.457ms/137.9Mbps	2.415ms/83.2Mbps	3.390ms/59.3Mbps	4.30ms/46.7Mbps

required to copy the inputs from host memory to GPU and copy the results on GPU back to the host memory.

An N-Way Parallel MIMO Detection requires N QR decompositions on an input $\mathbf{y}$ and $\mathbf{H}$. Each QR decomposition computes a unique QR by permuting the row of $\mathbf{y}$ and column $\mathbf{H}$. Table I shows the results for 2×2 and 4×4 MIMO configurations with different values of N. To ensure the WARP instruction is fully occupied, we pack up to 8 different QR decomposition problems in the same thread block for 2×2 and 4×4 MIMO configurations. We see that the QR decomposition kernel is not the bottleneck in the detection algorithm. For example, suppose QR decomposition is the dominate time for 4×4 64-QAM MIMO detection, the the worst scenario is processing 8400 MIMO symbols which takes 0.501 ms, which is equivalent to 384 Mbps.

Table II shows the results for different MIMO/Modulation configurations with different values of N. We pack up to 4 different detection problems in the same thread block for 16-QAM configurations. Since the number of threads required for the MIMO detector scales linearly with N, throughput is directly proportional to N. The bottleneck in the MIMO detection kernel for $M = 64$ is the hypothesis and counter-hypothesis generator which depends on modulation order M, but not the number of antenna. As a result, the runtime of the 2×2 64-QAM MIMO detector is not half of the 4×4 64-QAM MIMO detector. Since the number of bits transmitted is halved for 2×2, the throughput for 2×2 64-QAM is actually slower than 4×4 64-QAM.

We can compare the throughput performance for $N = 2$ and the results in [7] since the FER performance of these two detectors are similar. For 2×2 and 4×4 64-QAM, the Trellis MIMO detector achieved a throughput of 43.86Mbps and 12.05Mbps respectively. As shown in [7], given a node and M possible paths, the number of instructions required scales with M for [7], while design in this paper is a round and threshold function independent of M. As a result, the Trellis MIMO detector is much slower.

Table III shows the total runtime of the complete design measured with CPU timer. This includes overheads such as transport time which we found to be the dominant overhead. The design still remains faster than the design in [7].

VI. CONCLUSION

In this paper, we presented a high throughput GPU MIMO detector. We propose running multiple search blocks in parallel with different antenna detection order to scale performance. We show that by changing the number of search passes, we can increase or decrease accuracy to meet the requirements. We also show that even including overhead of QR decomposition, we can achieve a high performance detector on GPU.

REFERENCES

[1] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI Implementation of MIMO Detection Using the Sphere Decoding Algorithm," *IEEE J. Solid-State Circuit*, vol. 40, pp. 1566–1577, July 2005.

[2] K. Wong, C. Tsui, R. Cheng, and W. Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *IEEE Int. Symp. on Circuits and Syst.*, vol. 3, pp. 273–276, May 2002.

[3] M. Li, B. Bougard, E. Lopez, A. Bourdoux, D. Novo, L. Van Der Perre, and F. Catthoor, "Selective Spanning with Fast Enumeration: A Near Maximum-Likelihood MIMO Detector Designed for Parallel Programmable Baseband Architectures," in *ICC '08. IEEE International Conference on Communications*, May 2008.

[4] M. Wu, C. Dick, Y. Sun, and J. R. Cavallaro, "Improving MIMO Sphere Detection Through Antenna Detection Order Scheduling," in *SDR '11: Proceedings of the 2011 SDR Technical Conference and Product Exposition*, 2011.

[5] Q. Qi and C. Chakrabarti, "Parallel High Throughput Soft-output Sphere Decoder," in *IEEE Workshop on Signal Processing Systems (SiPS'10)*, Oct. 2010.

[6] T. Nylanden, J. Janhunen, O. Silven, and M. Juntti, "A GPU implementation for two MIMO-OFDM detectors," in *2010 International Conference on Embedded Computer Systems (SAMOS)*, pp. 293 –300, July 2010.

[7] M. Wu, Y. Sun, S. Gupta, and J. Cavallaro, "Implementation of a High Throughput Soft MIMO Detector on GPU," *Journal of Signal Processing Systems*, vol. 64, pp. 123–136, 2011.

[8] A. Kerr, D. Campbell, and M. Richards, "QR decomposition on GPUs," in *Proceedings of 2nd Workshop on GPGPU*, pp. 71–78, ACM, 2009.

[9] NVIDIA Corporation, *CUDA Compute Unified Device Architecture Programming Guide*, 2008.

[10] K. Amiri, C. Dick, R. Rao, and J. R. Cavallaro, "A High Throughput Configurable SDR Detector for Multi-user MIMO Wireless Systems," *Springer Journal of Signal Processing Systems*, vol. 62, pp. 233–245, February 2011.

[11] B. Hochwald and S. ten Brink, "Achieving Near-Capacity on a Multiple-Antenna Channel," *IEEE Tran. Commun.*, vol. 51, pp. 389–399, Mar. 2003.

[12] V. Volkov and J. W. Demmel, "Benchmarking GPUs to Tune Dense Linear Algebra," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1–11, 2008.