SOFT MIMO DETECTION ON GRAPHICS PROCESSING UNITS AND PERFORMANCE STUDY OF ITERATIVE MIMO DECODING

A Thesis

by

RICHEEK ARYA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2011

Major Subject: Computer Engineering

SOFT MIMO DETECTION ON GRAPHICS PROCESSING UNITS AND PERFORMANCE STUDY OF ITERATIVE MIMO DECODING

A Thesis

by

RICHEEK ARYA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Seong Gwan Choi
Committee Members,	Riccardo Bettati
	A. L. Narasimha Reddy
Head of Department,	Costas N. Georghiades

August 2011

Major Subject: Computer Engineering

ABSTRACT

Soft MIMO Detection on Graphics Processing Units and
Performance Study of Iterative MIMO Decoding. (August 2011)
Richeek Arya, B. Tech., International Institute of Information Technology, India,
Chair of Advisory Committee: Dr. Seong Gwan Choi

In this thesis we have presented an implementation of soft Multi Input Multi Output (MIMO) detection, single tree search algorithm on Graphics Processing Units (GPUs). We have compared its performance on different GPUs and a Central Processing Unit (CPU). We have also done a performance study of iterative decoding algorithms. We have shown that by increasing the number of outer iterations error rate performance can be further improved. GPUs are specialized devices specially designed to accelerate graphics processing. They are massively parallel devices which can run thousands of threads simultaneously. Because of their tremendous processing power there is an increasing interest in using them for scientific and general purpose computations. Hence companies like Nvidia, Advanced Micro Devices (AMD) etc. have started their support for General Purpose GPU (GPGPU) applications. Nvidia came up with Compute Unified Device Architecture (CUDA) to program its GPUs. Efforts are made to come up with a standard language for parallel computing that can be used across platforms. OpenCL is the first such language which is supported by all major GPU and CPU vendors.

MIMO detector has a high computational complexity. We have implemented a soft MIMO detector on GPUs and studied its throughput and latency performance. We have shown that a GPU can give throughput of up to 4Mbps for a soft detection algorithms which is more than sufficient for most general purpose tasks like voice communication etc. Compare to CPU a throughput increase of $\sim 7x$ is achieved.

We also compared the performances of two GPUs one with low computational power and one with high computational power. These comparisons shows effect of thread serialization on algorithms with the lower end GPU's execution time curve shows a slope of 1/2.

To further improve error rate performance iterative decoding techniques are employed where a feedback path is employed between detector and decoder. With an eye towards GPU implementation we have explored these algorithms. Better error rate performance however, comes at a price of higher power dissipation and more latency. By simulations we have shown that one can predict based on the Signal to Noise Ratio (SNR) values how many iterations need to be done before getting an acceptable Bit Error Rate (BER) and Frame Error Rate (FER) performance. Iterative decoding technique shows that a SNR gain of $\sim 1.5dB$ is achieved when number of outer iterations are increased from zero. To reduce the complexity one can adjust number of possible candidates the algorithm can generate. We showed that where a candidate list of 128 is not sufficient for acceptable error rate performance for a 4x4 MIMO system using 16-QAM modulation scheme, performances are comparable with the list size of 512 and 1024 respectively.

To My Family, Friends and Professors

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my advisor, Prof. Seong Gwan Choi, for his continuous support during my masters and for his patience, motivation and enthusiasm. He guided me through a whole new area of MIMO Communications, discussed new algorithms and gave vital feedback about my research. This thesis would not have been possible without his continuous guidance.

Besides my advisor, I would like to thank my committee members, Prof. Riccardo Bettati and Prof. A.L. Narasimha Reddy, for their continuous encouragement, insightful comments and questions. I would like to extend my sincere thanks to all the staff in the Department of Electrical and Computer Engineering especially Tammy, Carolyn and all my professors who made my academic life so easy and intellectually rewarding.

I am grateful to my friends Manish, Nitin and *Yo Buddy* group for their continuous assistance and encouragement during my tough times. I am thankful to my family, my *bua* Rekha and Raka, my *chacha* Manoj, my sisters Richa, Riddhi and my parents for their unconditional love and support all through my life.

Finally, I would like to acknowledge the help provided by my research group members Raj, Pankaj, Yoon and Ehsan whenever needed. Our discussions helped me to refine my work and get timely feedback on my approach.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

With continuous need of higher communication rate within a fixed frequency spectrum, Multiple Input Multiple Output technology along with Orthogonal Frequency Division Multiplexing (MIMO-OFDM) has provided a potential solution. Every new wireless communication protocol like WiMAX, Wi-Fi, LTE etc. is employing MIMO technology to satiate ever increasing demand of rate. In MIMO systems multiple antennas are employed on both transmitter and receiver side. Spatial multiplexing technique is applied to increase the rate. In spatial multiplexing higher rate input data stream is sub-divided into lower rate sub data streams. These data streams are then transmitted from multiple antennas. It has a potential of increasing the capacity N folds, where N is the minimum of the total number of transmit and receive antennas. On the receiver side received signal at each antenna corresponds to a combination of multiple data streams from all the transmit antennas. Following Figure 1 shows an example of spatial multiplexing. The input data stream with high data rate is split into multiple smaller data rate bit streams. These individual data streams are then further modulated in which a set of bits are assigned a fixed codeword in the symbol constellation. The advantages of MIMO systems are numerous. A typical MIMO system can be used to increase link reliability and Quality of Service (QoS) by using Diversity Gain methods. In this method the input stream is space-time coded and then transmitted which gives more robustness. MIMO system can also be used in multiplexing gain mode where each transmit antenna sends out independent bit streams hence throughput of the system is increased. Figure 2 summarizes MIMO

This thesis follows the style of *IEEE Antennas and Wireless Propagation Letters*.

use case scenarios.

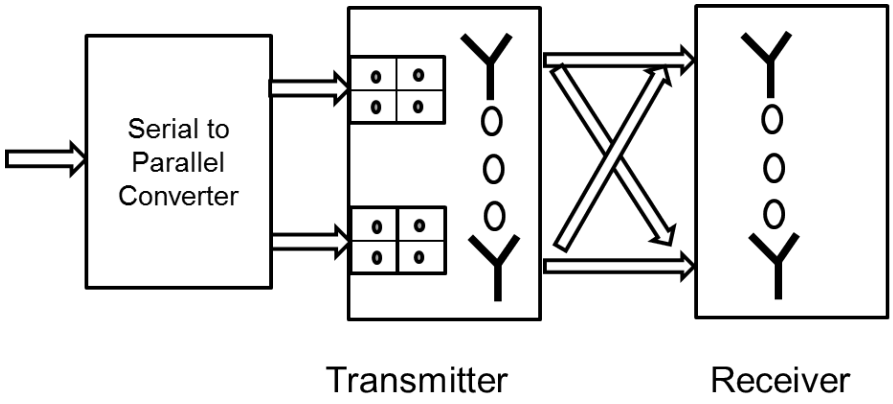

Fig. 1.: Spatial multiplexing

Diversity Gain	Multiplexing Gain
More Robustness	Multiplexed Data Streams
Link Reliability and QoS	Spectral Efficiency

Fig. 2.: Features of MIMO technology

Figure 3 shows a constellation diagram for a 16-QAM modulation scheme where each constellation symbol is a collection of 4 bits. On the receiver side each receive antenna receives a superposition of all the transmitted symbol vectors. The received symbols are shifted points in the constellation diagram (Figure 3) and its job is to remap them correctly to the sent points. To this end this becomes a search problem. For a Single Input Single Output (SISO) system which is using any Q^2 modulation scheme a simple brute force approach for any received symbold y receiver has to

search in Q^2 constellation points to find the nearest point.

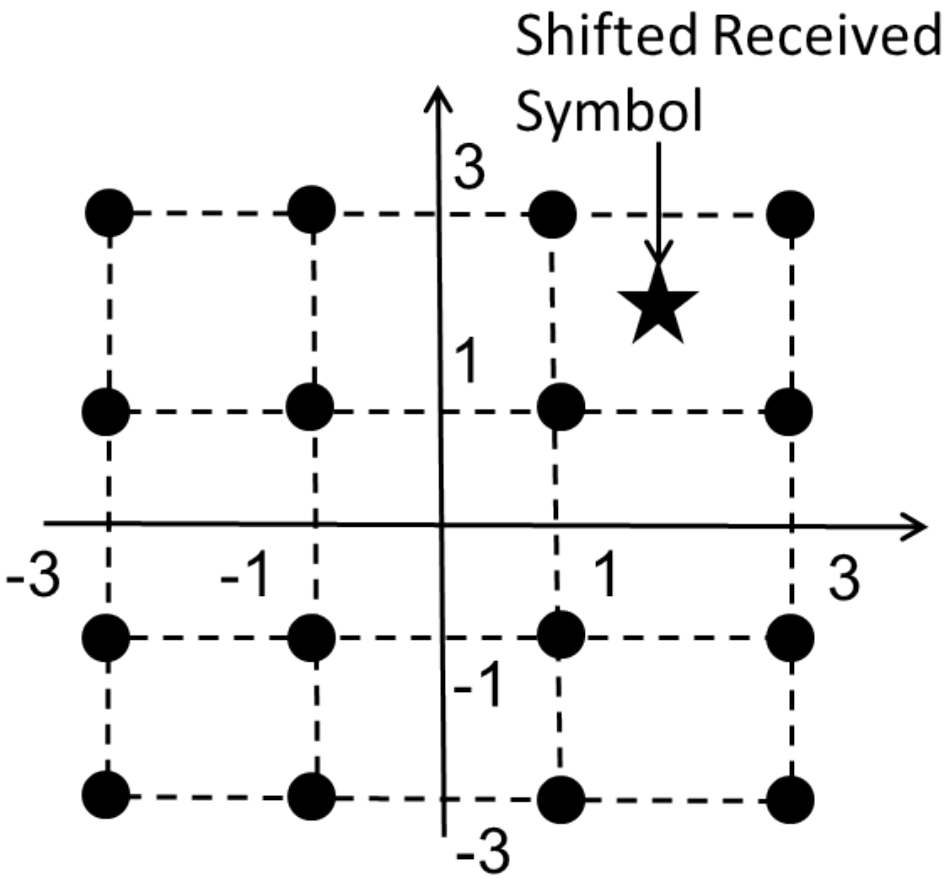

Fig. 3.: 16-QAM constellation diagram

However, as shown later in the discussion this task become exponentially complex if the system is a MIMO system. MIMO detection is a challenging task because of conflicting requirements of high bit rate and reduced implementation cost as shown in Figure 4.

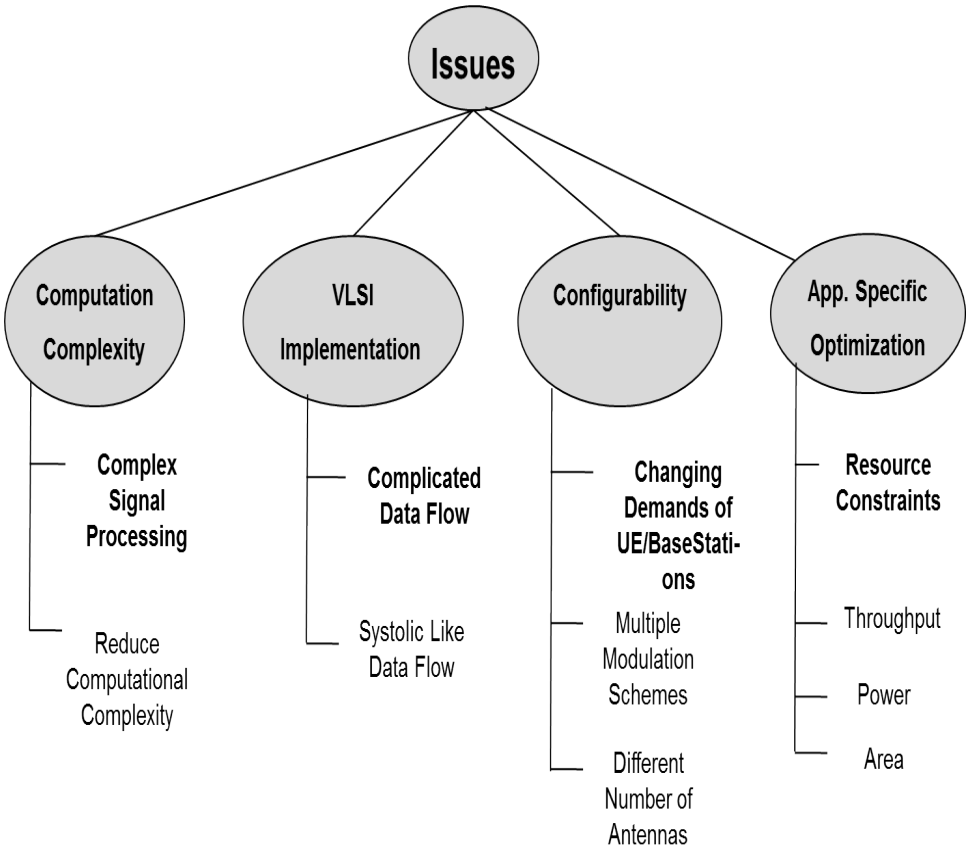

Fig. 4.: Challenges in MIMO detection

With reference to Figure 4 MIMO detection algorithms used in signal processing are more compute intensive. Since modern wireless standards pose throughput requirements of the order of Mega Bits per Second (Mbps), algorithms have to reduce computational complexity to achieve the target. Many MIMO detection algorithms are variable throughput algorithms. From a VLSI implementation perspective there are feedback paths which makes algorithm to have variable throughput. Hence to ease the implementation one wants to have a systolic like data flow. In systolic data flow all the feedback paths are removed and the algorithm can be efficiently pipelined. Modern wireless standards support multiple antennas at base station and user end. They also support multiple modulation schemes for data transmission. Hence a detec-

tor design should allow dynamic runtime configurability between different modulation schemes and different MIMO antenna configurations. With semiconductor industry's focus is on minimizing chip area and power consumption one has to keep a check on detector's area and dynamic and static power consumption numbers.

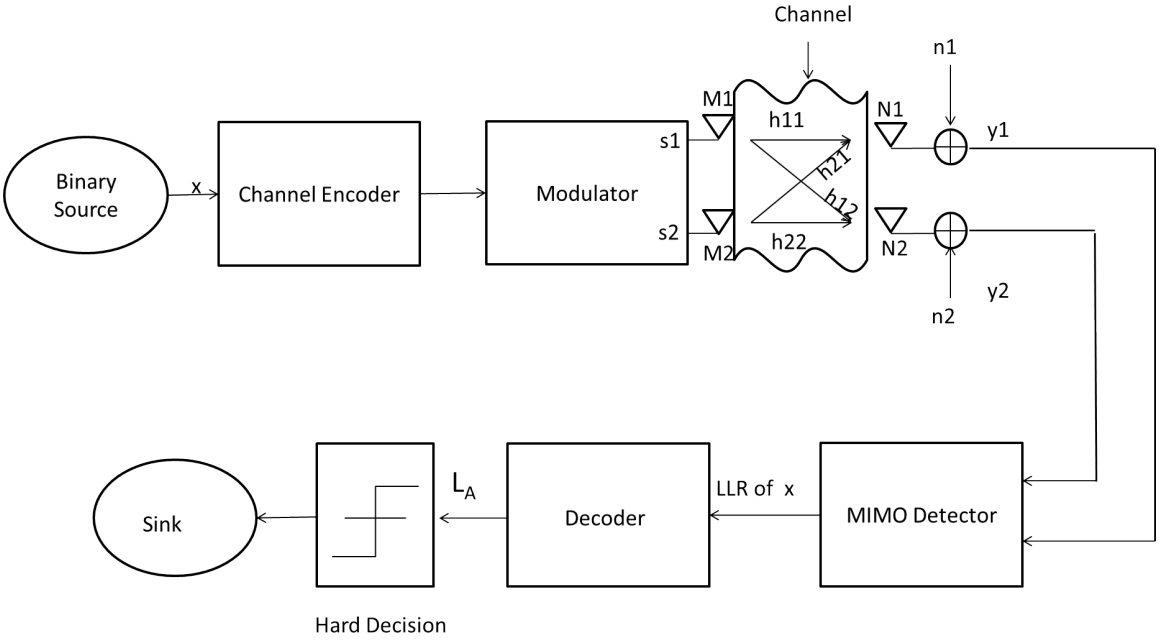

Fig. 5.: A MIMO system model

Figure 5 shows a basic 2x2 MIMO system with two transmit and two receive antennas. Binary source generates a stream of information bits x where $x \in \{0, 1\}$ which are to be transmitted over the wireless channel. These bits are first channel coded using an encoder (LDPC, Turbo Codes and Convolution Codes etc.). Encoded bit stream x is then mapped to constellation symbols (s_1 and s_2) called QAM symbols and transmitted. Each QAM symbol undergoes independent channel gains (h_{11}, h_{12} etc.) before reaching the receiver. These symbols are further corrupted by Additive White Gaussian Noise (AWGN) represented as (n_1, n_2). The group of QAM symbols

(s_1, s_2) is called a MIMO symbol.

Receiver receives a combination of all QAM symbols corrupted with noise. Job of the receiver is to correctly decode each QAM symbol. First the MIMO detector computes the estimate $\hat{s}$ of the most likely transmitted symbol sequence. These estimates are then fed into the decoder which gives out final decoded bits.

A. Baseband Representation of a MIMO System

An MxN MIMO system can be modeled as

$$y = Hs + n \tag{1.1}$$

where $y = [y_0, y_1, ..., y_{M-1}]$ is the received vector. H is the NxM channel matrix, where each element, $h_{i,j}$ is an independent and identically distributed zero mean circularly symmetric complex Gaussian random variable with unit variance. Noise at the receiver is $n = [n_0, n_1, ..., n_{N-1}]$, where n_i is an independent and identically distributed zero mean circularly symmetric complex Gaussian random variables with σ^2 variance per complex dimension. The transmit vector is $s = [s_0, s_1, ..., s_{N-1}]$, where s_i is drawn from a finite set of complex constellation alphabets, Ω, of cardinality Q. For example, the constellation diagram for 16 QAM is as shown in Figure 3 with cardinality Q = 16. In the next subsection I am going to briefly talk about different detection algorithms.

B. Different Detection Algorithms

Goal of any MIMO detection algorithm is to correctly map the received symbol on the constellation diagram. Figure 6 shows a SISO system where symbols were transmitted using QPSK modulation scheme. Due to independent channel gains and

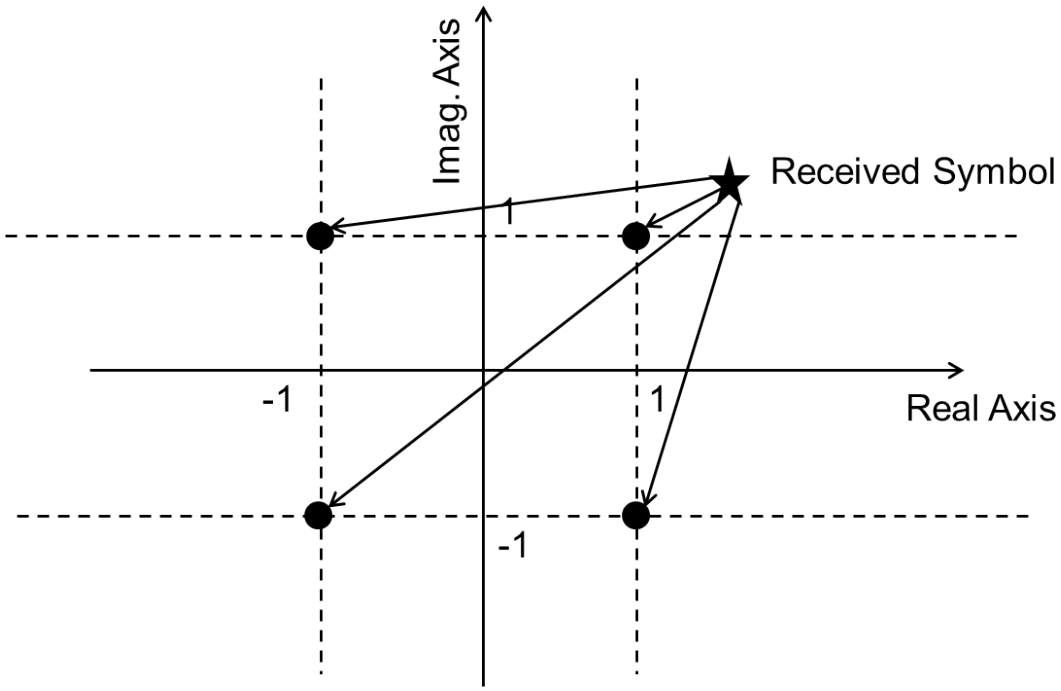

Fig. 6.: MIMO detection problem using a SISO QPSK example

noise (Equation (1.1)) the received symbol got displaced in the lattice as shown. Given the received symbol, job of a detection algorithm is to find which symbol was actually transmitted. To this end, it becomes a search problem in the lattice as shown in the Figure 6 the the symbol with least distance from the received symbol should be picked.

MIMO detection schemes can be broadly classified in three main categories based upon the underline schemes used for detection.

- Linear and Successive Interference Cancellation (SIC) detection
- Exhaustive Search and Maximum Likelihood Detection
- Iterative Tree Search Based Detection

Figure 7 shows the BER performance of these algorithms. It can be seen from the figure that ML based approach gives the best BER performance.

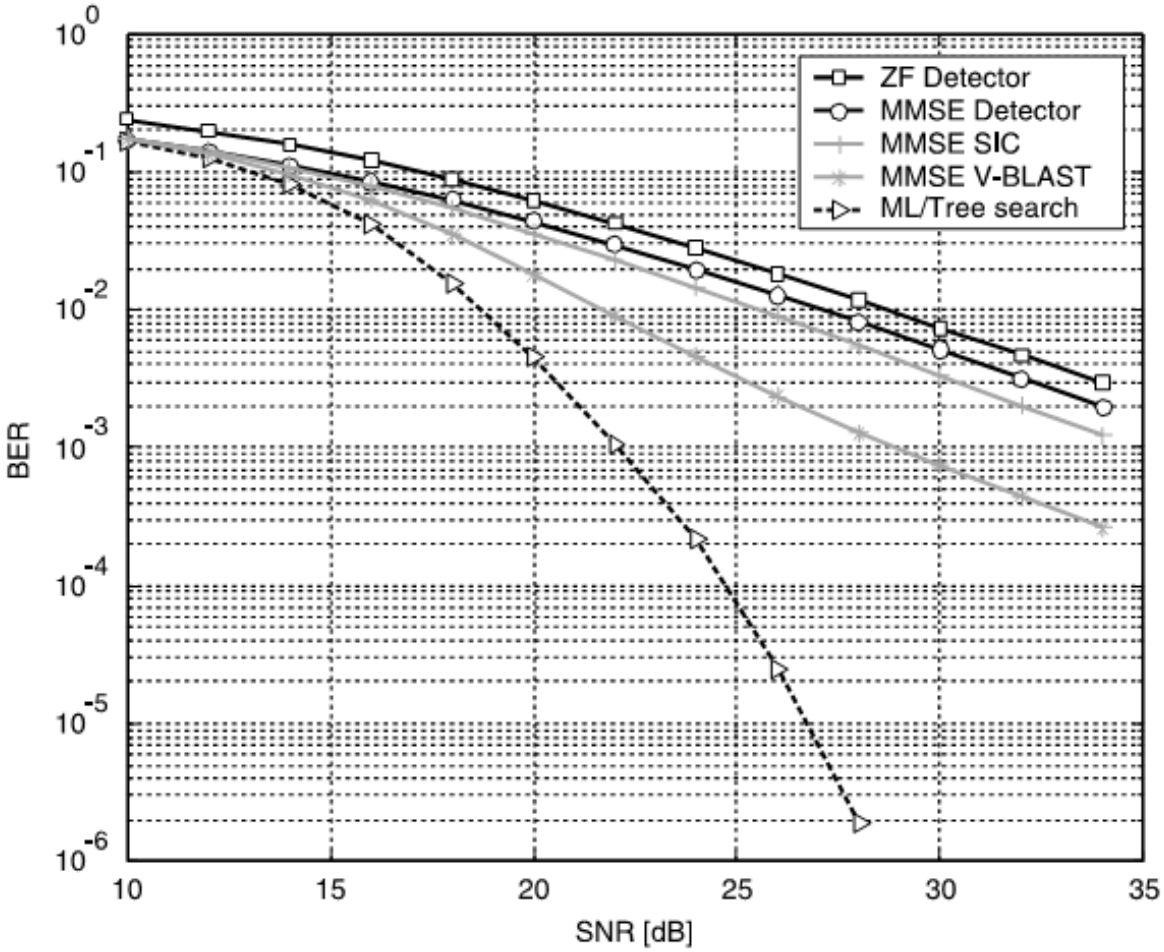

Fig. 7.: BER performance of different MIMO detection algorithms for 4x4 MIMO system using 16-QAM modulation scheme [1]

1. Linear and SIC Detection

Linear MIMO detection method treats the problem given by Equation (1.1) as an unconstrained Linear optimization problem. Zero forcing (ZF) method solves the

problem according to method of least squares. Other detector solves it using Minimum Mean Square Error (MMSE) criterion. They try to estimate the transmitted signal x from a received signal y by using the knowledge of channel matrix H as shown in equation (1.2).

$$\hat{s} = \hat{W}y \tag{1.2}$$

W is the channel estimator matrix. $\hat{s}$ does not consider the fact that vector s was chosen from a limited set of constellation points O. Hence an additional step called slicing is performed on each entry of s to map it to the nearest constellation point.

$$\hat{x_i} = map(\hat{s_i}) \tag{1.3}$$

However, the main drawback of these linear estimation algorithms is that they can only achieve a diversity order of $N - M - 1$ which become clear in systems with equal number of transmit and receive antennas as shown in the following Figure 7.

Zero forcing detector inverts the frequency response of the channel. W for zero-forcing detector is the pseudo-inverse of H. For symmetric systems it is same as H^{-1}. Applying this on the received vector y yields Equation (1.4).

$$\hat{s} = s + n_{ZF} \tag{1.4}$$

where $n_{ZF} = H^{-1}n$. Although Zero forcing detector has removed the interference between parallel streams but noise power is now increased which degrades the performance. The second type of linear detector is called Minimum Mean Squared Error (MMSE) detector. MMSE detector instead of forcing parallel interfering streams to zero, tries to minimize overall expected error in detection. It takes channel noise into

account. It can be shown [1] that channel estimator matrix W can be given as:

$$W = (H^H H + M\sigma^2 I)^{-1} H^H \tag{1.5}$$

where H^H is the Hermitian of H.

Successive Interference Cancellation Scheme is also a linear estimation algorithm. It partially exploits the fact that transmitted symbol s has been chosen from a finite set Ω of constellation points. Hence instead of detecting all symbols at once, they are considered one after another and their effect is subtracted from subsequent streams. In comparison with other linear detection algorithms SIC achieve higher diversity order with each next detected stream. However, the overall BER performance is dominated by the stream detected first and error propagation also plays a dominant role in the performance. Hence detection order is of high importance here to achieve good BER performance.

2. Maximum Likelihood Based Detection

The objective of a MIMO detector is to estimate $\hat{s}$ based on the observation y along with the knowledge of channel matrix H. The optimal or the Maximum Likelihood (ML) estimate $\hat{s}_{ml}$ is given by the equation (1.6):

$$\hat{s}_{ml} = \min_{s \in \Omega^M} ||y - Hs||^2 \tag{1.6}$$

However, to achieve an optimum BER performance the algorithm's complexity grows exponentially in terms of number of antennas and constellation size. To get an idea, a direct implementation of the above algorithm will lead to searching all the possible versions of s and select the one for which equation (1.6) is minimum. This direct approach becomes prohibitively expensive as more number of antennas and/or higher

modulation schemes are used. For example in a 4x4 MIMO system using QPSK (2 bits per QAM symbol) and 16-QAM (4 bits per QAM symbol) total number of MIMO symbols to search through will be $4^4 = 256$ and $16^4 = 65{,}536$ respectively.

3. Tree Search Algorithms

The basic idea behind iterative tree-search algorithms lie in transformation of the original ML detection problem into a fully optimal or sub-optimal ML detection problem. ML detection is now formulated into an equivalent tree search problem in which distances between received vector y and the received candidate symbols Hs can be decomposed into partial Euclidean distances (PEDs) $d_i(s^{(i)})$ which depend only on $s^{(i)}$ and which is a non-decreasing non-negative function when proceeding from a parent node to its children. Based on these PEDs, tree-search algorithms aim at finding the leaf that is associated with the smallest $d_i(s^{(1)})$ which corresponds to the ML solution. To create a tree structure first QR decomposition is applied on H matrix which converts it into a product of upper triangular matrix R and a unitary matrix Q. Equation (1.6) can now be re-written as:

$$\begin{aligned} \hat{y} =& Rs + Q^H n \\ \hat{y} =& Q^H y \end{aligned} \tag{1.7}$$

where $H = QR$ The original detction problem shown in (1.6) is now remodeled as shown in Figure 8. Due to the triangular structure of R the partial distance of i^{th} QAM symbol $s^{(i)}$ is only a function of QAM symbols ($s^{(i+1)}$, $s^{(i+2)}$,..., $s^{(M)}$) hence the original problem is transformed to a tree structure. Figure 9 illustrates this using a BPSK modulation example. Each level of the tree corresponds to a transmit antenna. At each level we enumerate all possible QAM symbols corresponding to modulation scheme used for that particular antenna. Since for a BPSK modulation scheme there

are only two possible QAM symbols hence every parent has two children. Each node's distance only depends on its parent's distance and itself. Therefore, $s^{(3)}$ depends only on root node and $s^{(3)}$. $s^{(2)}$ on its parent and $s^{(2)}$ and so on. Hence with reference to Figure 9 we can now write PEDs of nodes by equation 1.8.

$$\begin{aligned} d_i(s^{(i)}) =& d_{i+1}(s^{i+1}) + |b_{i+1}(s^{i+1}) - R_{i,i}s_i|^2 \\ b_{i+1}(s^{i+1}) =& y_i - \sum_{j=i+1}^{M} R_{i,j}s_j \end{aligned} \tag{1.8}$$

Please note that equation (1.8) represent a recursion. For a symmetric system ($M = N$) the start condition of the recursion is $d_{M+1}(s^{M+1}) = 0$.

$$d(\mathbf{s}) = \left\| \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} - \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} \right\|^2 \Leftrightarrow \left\| \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix} - \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} \right\|^2$$

Fig. 8.: Effect of QR decomposition

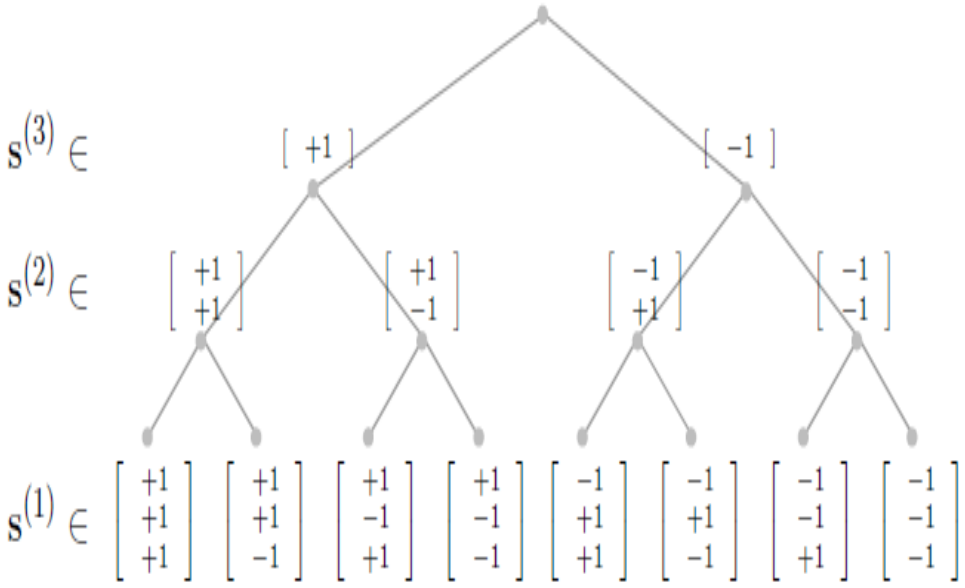

Fig. 9.: A BPSK MIMO example with 3 transmit antennas [1]

C. Tree Traversal Methods

- Depth First Search(DFS)
- Breadth First Search(BFS)

DFS is a recursive scheme. The search starts from the root node ($i = M + 1$) and explores down as deep as possible the tree until it hits a leaf node. At this point the algorithm backtracks and goes to the parent and search its other child. BFS starts from the root node and for any node it first traverse all its siblings before moving to their children. Hence the algorithm moves only in forward direction in a systolic fashion. One of the famous detection algorithms Sphere decoding uses a DFS Model and achieves an optimal BER performance whereas K-best algorithm uses BFS and it is a sub-optimal search algorithm.

1. Sphere Decoding (SD)

Sphere decoding algorithm is a type of tree search algorithm. Key idea here is to reduce the number of candidate symbol vectors to be considered for the ML solution. Search includes only those candidate symbol vectors s that lie inside an hypersphere of radius r (1.9).

$$||y - Hs||^2 < r^2 \tag{1.9}$$

This is known as sphere constraint (SC). Since it has been stated the PEDs are non-decreasing functions hence with reference to equation (1.9) it means to visit only those nodes for which the following inequality holds:

$$d_i(s^i) < r^2 \tag{1.10}$$

If for any node equation (1.10) does not satisfy then one can prune the tree originating from that node without sustaining any performance penalty.

An important aspect that one needs to consider is how to select an initial radius r to start with. One way is to start with an initial guess and dynamically update it as one finds a leaf node with smaller PED than SC. Another way is to start with a zero-forcing or some other educated guess and iteratively update the radius. Choice of radius will directly results in how fast the algorithm will converge to a solution. If the radius is large many nodes will satisfy the SC and search will take more time to converge. However, if the chosen radius is too small then no node may fit in and no solution will be found. Hence throughput of this algorithm is not fixed.

2. K-Best Decoding

In contrast to SD algorithm, K-best is an approximation algorithm which does not require a SC. Instead, tree pruning is enabled by constraining the number of admissible nodes at each level of the tree to a parameter K. Hence at each level out of all the nodes only K nodes with least PEDs are selected as admissible children. The choice of K will have a direct consequence on the decoder complexity. A large value of the K would mean more nodes to be visited whereas a small value may result in loss of optimal solution. However, unlike SD it has a fixed throughput since number of nodes traversed at every level is the same.

3. Tree Pruning

Tree pruning strategies are used to reduce the complexity of the tree search based algorithms. The basic idea is to reduce the number of tree nodes visited to achieve a ML solution. The decision where to visit a node or prune it is based on its PED. Depending upon tree pruning strategy the algorithm may or may not achieve optimal BER performance. Figure 10 shows the effect of pruning with initial SC is set to ∞. Once a leaf node with weight less than SC is found, SC constraint is updated. Nodes with weight greater than current SC are then pruned (shown with dashed arrows).

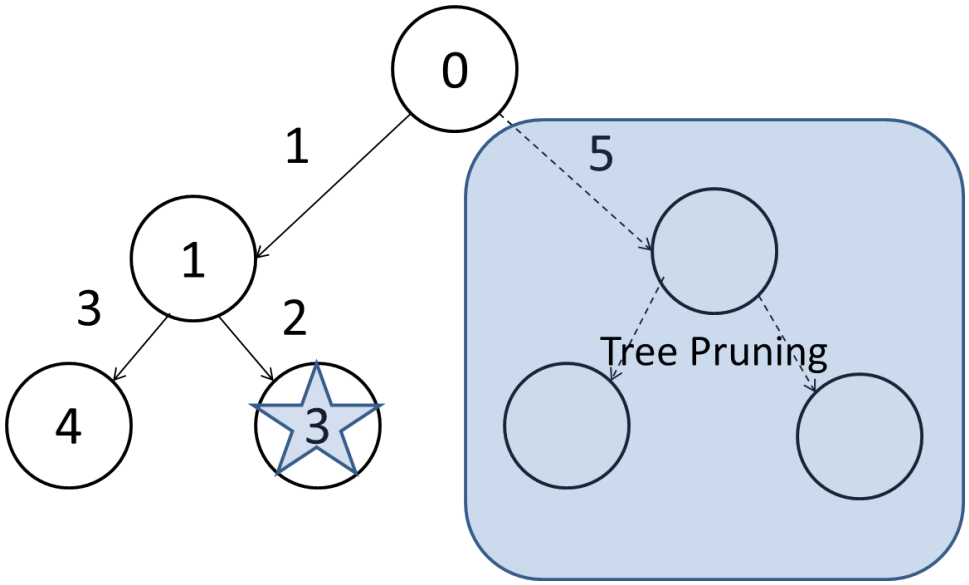

Fig. 10.: Tree pruning example

CHAPTER II

BACKGROUND

A. OpenCL Introduction

OpenCL is the first open parallel programming language for programming CPUs and GPUs [2]. It is cross platform and royalty free. The advantage of programming in OpenCL is that one can run the same code across different hardware platforms with virtually no change in it. All major vendors in Semiconductor industry like AMD, NVIDIA and Intel are supporting OpenCL in their latest CPU, GPU and System on Chip (SOC) products [2]. The advantage of OpenCL is that one can have different hardware choices from different vendors to choose from and he/she can compare the performance of different systems without changing the code much. I will briefly describe AMDs implementation of OpenCL on its Single Instruction Multiple Data (SIMD) hardware. SIMD means that the system is running one instruction simultaneously on multiple data streams. For example adding two arrays like $c[i] = a[i] + b[i]$ is a type of SIMD instruction if the same addition runs on different threads with different array indexes. In our perspective these SIMD engines are nothing but Graphics Processing Units. NVIDIA's implementation of OpenCL is similar to its CUDA implementation since it maps OpenCL kernels to CUDA kernels.

1. OpenCL Programming Model

In openCL implementation there is a host device and several other devices connected to it by a bus (like PCIe). These device are programmed using OpenCL C which is a C like language. It conforms to C99 programming standards, with an additional support for parallel programming such as memory fence operations and

barriers. Figure 11 shows OpenCL host/device architecture with a single platform consisting of a CPU and a GPU device.

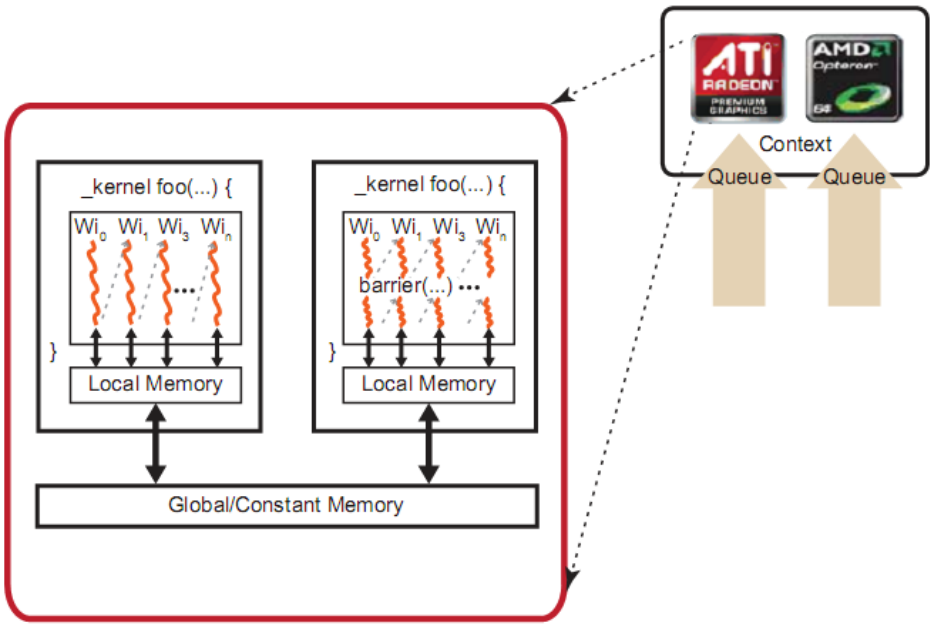

Fig. 11.: OpenCL Programming Model [3]. Commands are enqueued on a device. Each threads runs a copy of kernel.

Now I will present some definitions [4] that will make it easy to understand the following discussion. A device comprises of multiple compute units. A kernel is a piece of code that runs on an OpenCL device. A kernel code is identified with __kernel identifier. A command queue is used to enqueue commands on a device. A context is an abstract environment in which the kernels execute. Memory management is defined within the context. A host is defined as an entity which interact with the context using OpenCL API. A command queue is an object that stores commands to be executed on a device.

The general pattern for programming in OpenCL is as follows:

- Get the platform information which gives information about the vendors of the

CPU and the GPU present on the system

- Given the platform information select a device to create a context, allocate memory, create a command queue and perform data transfers
- Write kernel to execute on the selected device and submit it to the device's command queue
- Read data back to the host from the device

In the subsequent discussion all terms within the parenthesis denotes NVIDIA's terminology used in CUDA and outside of parenthesis represents OpenCL's generic terminology. OpenCL maps the total number of work-items (threads) to be launched onto an n-dimensional grid NDRange (grid). To provide data parallelism developer can further partition these work items in to work-groups (blocks). AMD GPUs execute on wavefronts (warp). Wavefront execute N number of work items in parallel and there are an integer number of wavefronts in each work-group. For ATI Radeon 5870 card wavefront size is 64. This is the lowest level of SIMD execution. All the work items within a wavefront execute same instruction. Hence this is also the least level at which flow control can affect. This is an important consideration in parallelism since if two work items go to divergent paths then all work items within a wavefront will execute both the paths hence execution time will become twice. Figure 12, shows the grouping of work items.

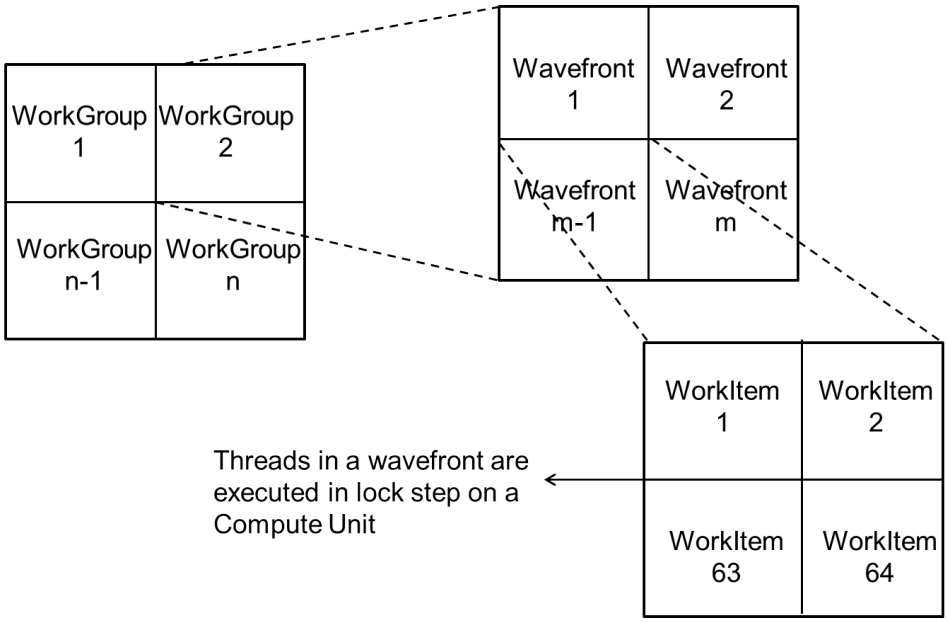

Fig. 12.: Work item, work group and NDRange grouping

OpenCL uses a shared memory model. Memory hierarchy in OpenCL is shown in following Figure 13. OpenCL uses a relaxed consistency memory model which means two threads accessing same memory location may get different values until a barrier or memory fence is applied. Private memory is accessible only within a work item. Local memory is accessible within a work group. Global memory is accessible to all work items in all work groups and Constant memory is a read only global space (cached)

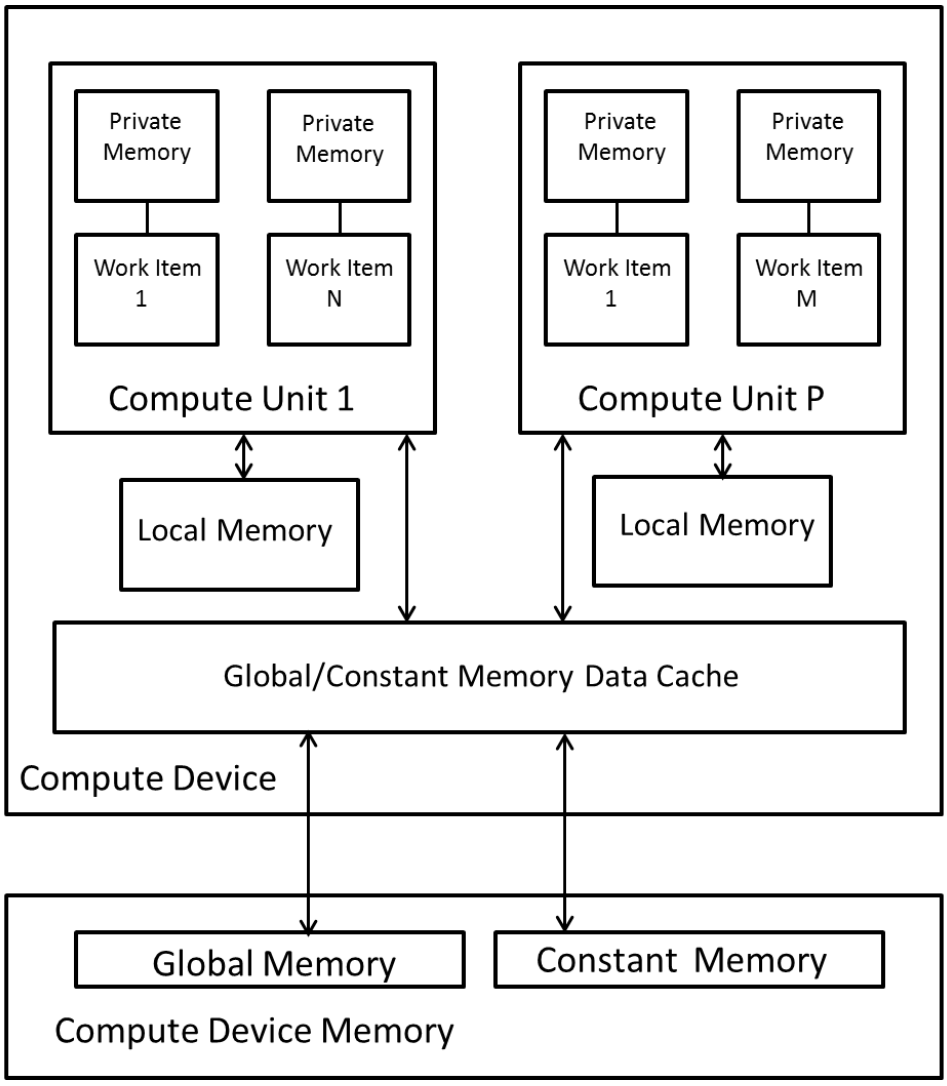

Fig. 13.: OpenCL memory model [4]

2. AMD's Graphics Processing Unit (GPU)

Different GPUs have different characteristics (such as the number of compute units), different amount of memory, clock speed but they follow a similar design pattern. GPU's compute devices comprise of groups of compute units. Figure 14 shows typical GPU organization: Within a compute unit there are numerous stream cores which execute kernels. Each stream core operates on an independent data stream.

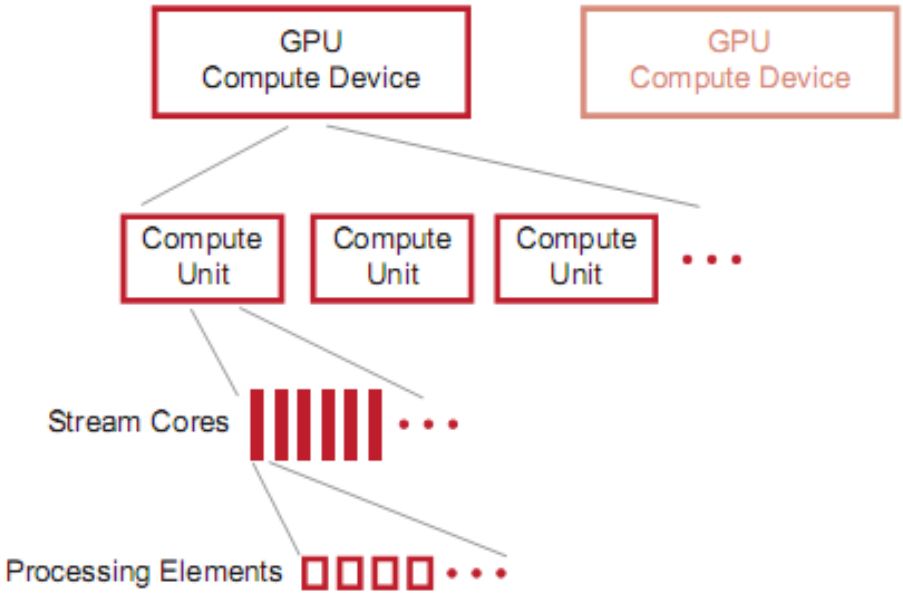

Fig. 14.: A typical GPU organization [3]

Stream cores are further divided into processing elements which are fundamental programmable processing units and carry out single precision, double precision operations. Figure 15 shows an AMD's compute unit. All stream cores within a compute unit execute the same instruction sequence; different compute units can execute different instructions.

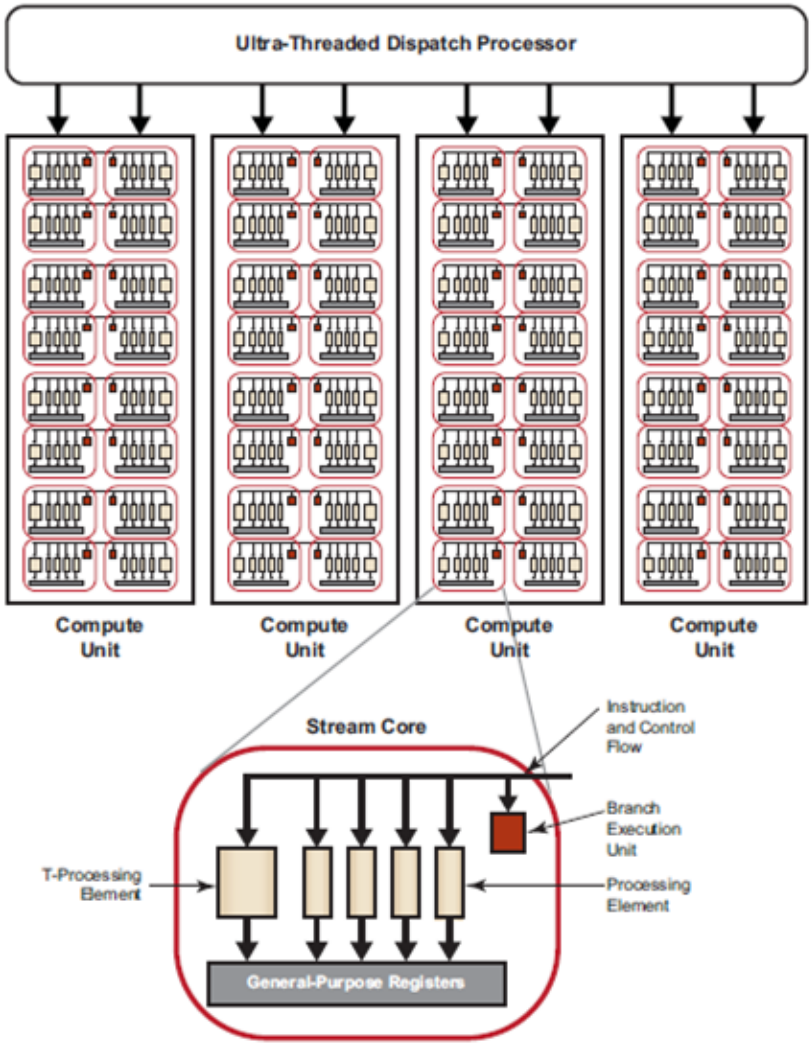

Fig. 15.: A typical compute unit [3]

B. Soft MIMO Detection

So far we have talked about detection of un-coded bit stream hence the output of the detector is a hard decision for each bit. However with reference to Figure 5 bits transmitted are generally encoded using some encoding scheme like LDPC. Hence to decode these bits detector generates soft values (bit probabilities) that can be processed by LDPC decoder. These probabilities are called a posteriori Log Likelihood

Ratios (LLRs) or maximum a posteriori probabilities (MAP) for each bit. These LLRs are also known as extrinsic LLRs (L_E) and can be defined as:

$$L(x_i|y) = \ln \frac{P(x_i = 1|y)}{P(x_i = 0|y)} \tag{2.1}$$

Where x_i is the i^{th} bit of the binary vector x and y is the received symbol. The same tree based search methodology can be used here to generate soft values with some modifications. Computation of equation (2.1) for each bit can be approximated using Max-log approximation [5] as:

$$\begin{aligned} L(x_i|y) \approx & \min_{X_{i,0}}(\Gamma(x,y)) - \min_{X_{i,1}}(\Gamma(x,y)) \\ \Gamma(x,y) = & ||y - Hs||^2 \\ s = & map(x) \end{aligned} \tag{2.2}$$

where $X_{i,0}$ and $X_{i,1}$ are two disjoint sets of bit vector x with i^{th} bit 0 and 1 respectively. $map(x)$ is the mapping of bit vector x in the constellation. One of the two terms in the equation (2.2) corresponds to the ML solution since a bit would be either 1 or 0 in the original sent sequence. However, calculation of the second term for each bit will become exponential in complexity. Consider the following scenario: In any Q^2-modulation scheme with number of transmit antennas M, total number of bits transferred are MQ. There are 2^{MQ} total number of unique combinations of these bits. To calculate any term of equation (2.2) one has to consider 2^{MQ-1} possible combinations for each bit. One can see from here that as complexity of MIMO systems grow large (by adding more number of antennas or higher modulation schemes), calculating LLRs become prohibitively expensive. Techniques discussed so far viz. sphere decoder and K-best decoder would only provide the ML solution but not the other term.

1. Repeated Tree Search (RTS)

In [6] authors have shown that (2.2) can be solved by first running SD algorithm to get the ML solution term of Equation (2.2) and then rerun the SD for each of the MQ bits with one bit flipped at a time. In the present example we are using a BPSK modulation scheme where each QAM symbol corresponds to a single bit. Hence we will use bit and QAM symbol interchangeably. However, in higher modulation schemes like 16-QAM each QAM symbol corresponds to 4 bits. Suppose transmitted MIMO symbol was [1 0] (bit 2 = 1 and bit 1 = 0). As shown in Figure 16, first search is performed to detect the ML symbol. Leaf nodes corresponds to level 1. Corresponding to bit 2 we enumerate all possible transmitted QAM symbols at level 2. Since transmitted QAM symbol was 1 hence PED of right child of root node has lesser distance than left child. So we select the right child. At the right child we again have two possibilities for bit 1. Using the same argument as before now the left child will have less distance than right one, so we pick the left child. Highlighted nodes corresponds to the detected symbol.

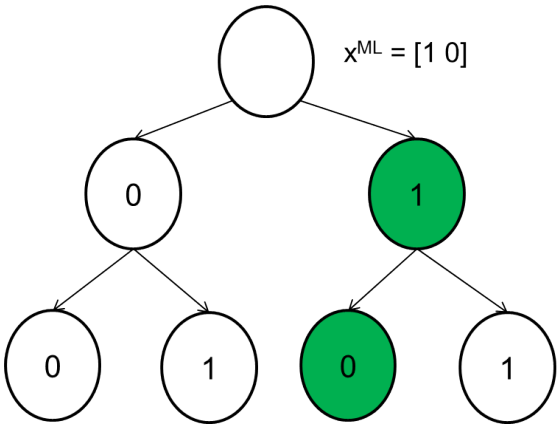

Fig. 16.: RTS: ML solution for a 2x2 system using BPSK modulation for transmitted symbol x = [1 0]

We run RTS again with each bit of the MIMO symbol flipped one at a time. Figure 17 (a) shows the search with bit 2 flipped. Nodes traversed are shown highlighted. In the final solution level 2 left node and level 1 right node are selected. Again bit 1 is flipped and tree is traversed again as shown in Figure 17 (b) with nodes visited highlighted. It is clearly visible from Figure 17 (a) and (b) that some of the nodes of tree are visited twice which is undesirable. RTS approach makes repeated traversal of large part of the tree and does redundant calculations multiple times as shown in Figure 17. In [7] authors have provided another approach called Single Tree Search in which the goal is to traverse each node at most once. respectively.

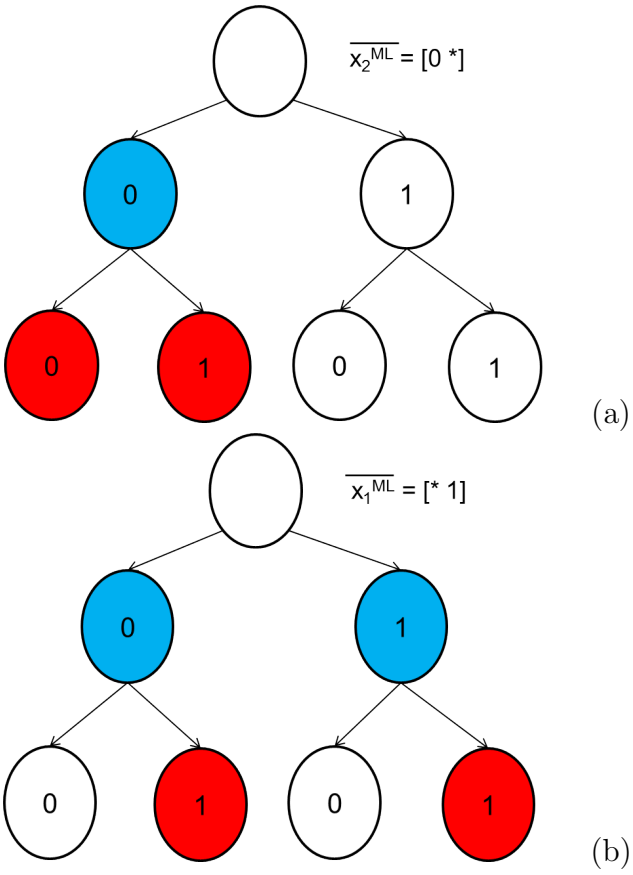

Fig. 17.: RTS for (a) bit 1 and (b) bi2 respectively for 2x2 System using BPSK modulation

2. Single Tree Search (STS)

Main objective is to traverse each node at most once. Tree is only pruned when the node's distance is greater than pruning radius and radiuses of all bits' counter hypothesis. Compare to RTS it reduces the complexity by a factor of 4 to 8. STS is one of the most efficient algorithms for soft MIMO detection. Counter hypothesis [7] of a bit is defined as $\bar{x}_j^{ML}$ which is the binary complement of the j^{th} bit in the ML solution. The idea of STS can be best explained with the aid of Figure 18. It sets distance of ML solution and counter hypothesis $\bar{SC}^{ML}$ of each bit to ∞. It then checks if the present node is a leaf node or not. If the present node is not a leaf node then it is pruned only if its PED is greater than SC and counter hypothesis of all the bits. Which is logical since we already have a better solution than this. If the present node is a leaf node then ML solution and counter hypothesis are modified accordingly.

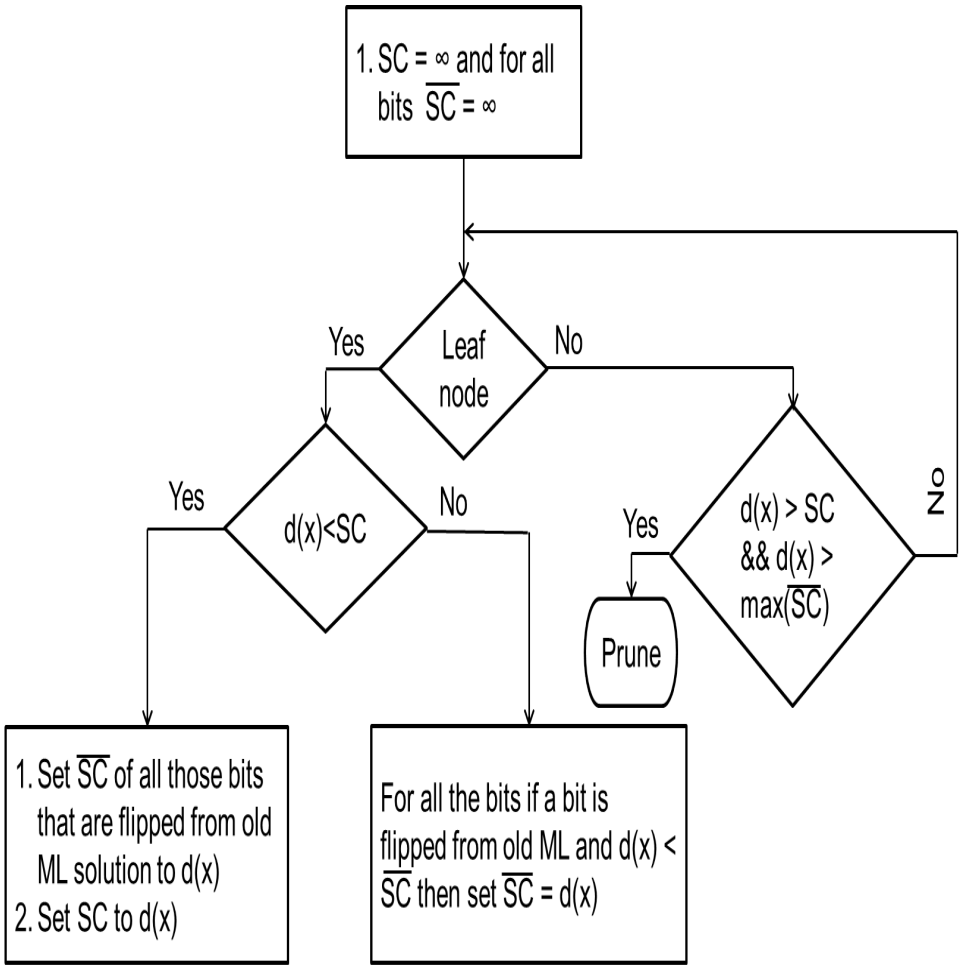

Fig. 18.: Flowchart for single tree search algorithm

In [8], authors have shown an implementation of soft detection algorithm on a high performance NVIDIA's GPU GeForce GTX 285, which shows a consistent linear speed up compare to the single threaded CPU run. However, running the algorithm without any optimizations limits the performance.

3. List Sphere Decoder

In [9] authors have proposed List Sphere Decoder (LSD) which simply modifies the sphere detector to generate a candidate list L of N_{cand} that makes each term of

equation (2.2) smallest. The list, by definition must include the ML solution. The size of the list obey $1 \leq N_{cand} < 2^{M.Q}$. Although the size of the list is chosen sufficiently large to include the counter hypothesis for each term but in cases when the list size is not large enough a clipping value can be used to approximate the term. There are couple of changes in the original sphere detection algorithm. Whenever a new candidate is found which is inside the SC

- It does not decrease r to update the SC
- It adds this point in the list if it is not full. If the list is full it replaces the point with the largest radius in the list with this one.

Apart from LSD there are several other SISO algorithms discovered that are used in MIMO. Algorithms proposed in [10, 11] are related to LSD but they rebuild the candidate list in each iterations and hence are computationally more expensive than LSD. [12, 13] require multiple matrix inversions at each iteration for each symbol vector. Compare to these algorithms QR Decomposition is only required when channel state changes.

CHAPTER III

GPU IMPLEMENTATION OF SINGLE TREE SEARCH AND PERFORMANCE STUDY OF ITERATIVE DECODING

A. GPU Implementation of STS

We envisage systems running baseband processing on GPUs. As shown in the figure on page 35 we need three main units to build the receiver, a soft detector, a LLR update unit and a decoder. We are going to present implementation results of soft MIMO detector on GPU. As shown in [7] the Single Tree Search (STS) sphere detector is an efficient implementation which minimizes node traversals and still gives better performance than its counterpart List Sphere Detector (LSD). Hence we have considered STS for GPU implementation.

In this subsection we have shown the implementation of STS algorithm on an ATI Radeon 5450 and a NVIDIA GTX 260 graphics processor. We have used OpenCL to program ATI graphics card and CUDA for GTX 260. The focus of the work is not to fine tune the algorithm for one particular GPU but to make it portable to any GPU with as little effort as possible.

GPU gives best performance when there is no flow control in the algorithm. If one has branches in the code then instead of all the threads running concurrently GPU will serialize them which essentially increases the execution time and decreases the throughput. STS algorithm is not free from flow control and some serialization of threads do occur here too. However, since there are 27 Compute Units in GTX 260 multiple sphere detectors are still running concurrently. So we see a throughput increase of about 7x at higher block sizes. Since each compute unit execute independently of other [3] the more threads one can feed in the better one can hide the

latencies. Hence we see a continuous increase in GPU's throughput until the block size reaches a threshold of about $1.4x10^5$ bits at which point feeding more threads does not increase the throughput. If we compare this throughput with the throughput we have achieved with ATI's lower end graphics card it becomes clear that having just 2 Compute units is not sufficient for an algorithm that has multiple divergent paths to attain sufficient throughput.

1. Simulation Results

We used Visual Studio 2010 Professional, with Matlab R2010a for simulations. To interface CUDA with Matlab we used Nvmex script [14]. The first system has NVIDIA GTX 260 that has 27 Compute Units and an Intel Core i7 CPU with 8 cores. The second system has an ATI Radeon 5450 card with 2 Compute Units and an AMD Athlon Quad core processor. We have simulated a 4x4 MIMO system where each antenna is using QPSK modulation scheme. Simulation is averaged over 100 channel realizations and CPU version of the code is running a single threaded application.

Figure 19 compares the throughput of CPU with GTX 260. As shown in the Figure 19 CPU throughput remains constant at about 0.6Mbps whereas as the block size increases GPU throughput rises continuously till about 4Mbps at which point it saturates. This can be explained since as we add more and more threads in the system the Compute Unit utilization increase till the point it cannot increase anymore at which point additional threads have to wait outside the system.

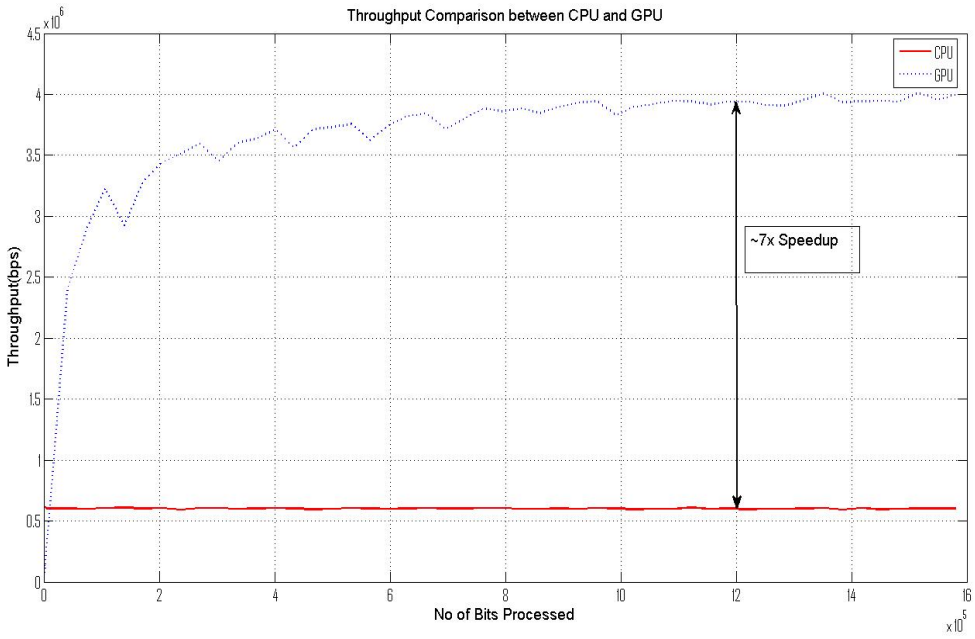

Fig. 19.: Throughput comparison of CPU vs GPU (GTX 260) which shows a speedup of $\sim 7x$

If we compare the execution time of GTX 260 with CPU as shown in Figure 20, the slope of the CPU curve is almost 1 which shows that as one increases the number of threads, execution time increases proportionally. However, in the case of GPU slope is less than 1 which clearly indicates that multiple threads are running concurrently.

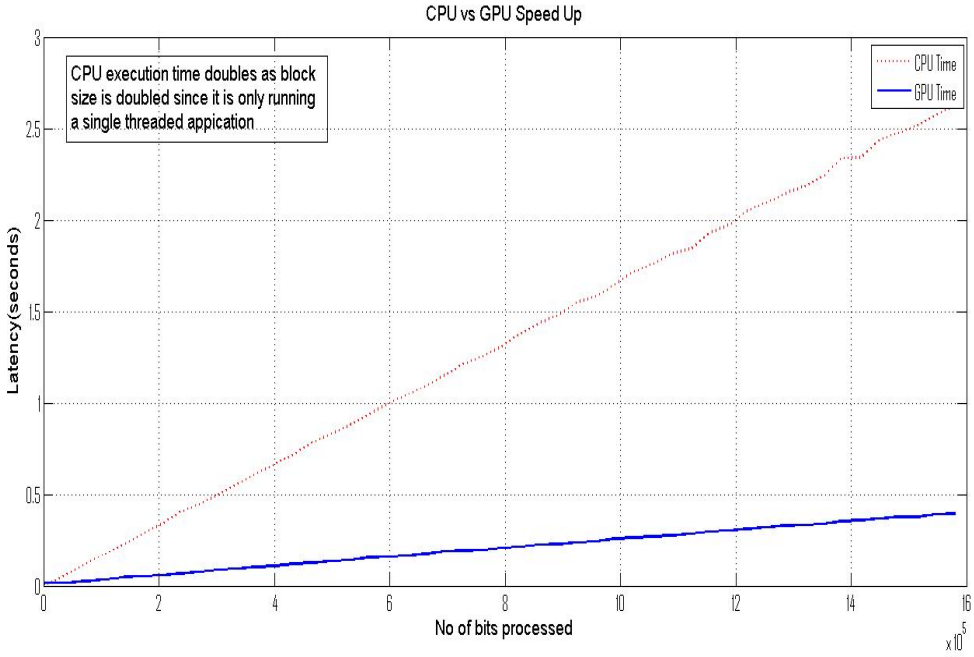

Fig. 20.: Execution time comparison of CPU vs GPU (GTX 260)

Performance comparison between two GPUs shows an interesting result. Figure 21 compares execution time of Radeon 5450 with GTX 260. Since Radeon 5450 has only 2 compute units when we plotted its execution time latency with number of bits processed we saw a slope of roughly 1/2, which is equal to inverse of number of compute units the GPU has. This can be explained since there is some flow control within the algorithm, serialization of threads is happening. However, since there are 2 compute units and each compute unit runs independently of other at any point of time at least 2 threads are running concurrently.

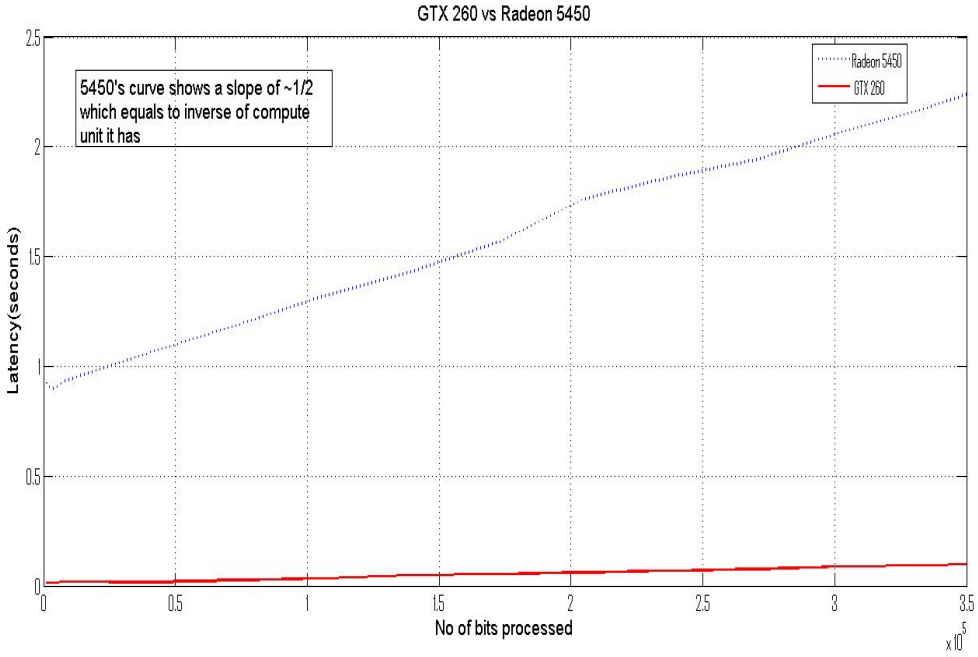

Fig. 21.: Execution time comparison between GTX 260 and Radeon 5450

B. Iterative Detection and Decoding

For the MIMO system shown in Figure 5 the soft output of the detector is fed to the decoder which finally outputs decoded bits. The detector generate extrinsic LLRs which are fed to the decoder. Decoder does channel decoding and generate a priori LLRs for each bit. A hard decision is taken based on a priori LLRs. An important aspect that one can explore is how much of this soft a priori information can be fed back to the MIMO detector to increase the overall error rate performance of the system. Researchers in [9] have shown that several dBs of SNR gain can be achieved using iterative methods. This type of decoding is called iterative detection decoding or just iterative decoding. Figure 22 shows the iterative decoding model. The system is almost same as Figure 5 with an exception of a LLR update unit and a feedback path from decoder to detector.

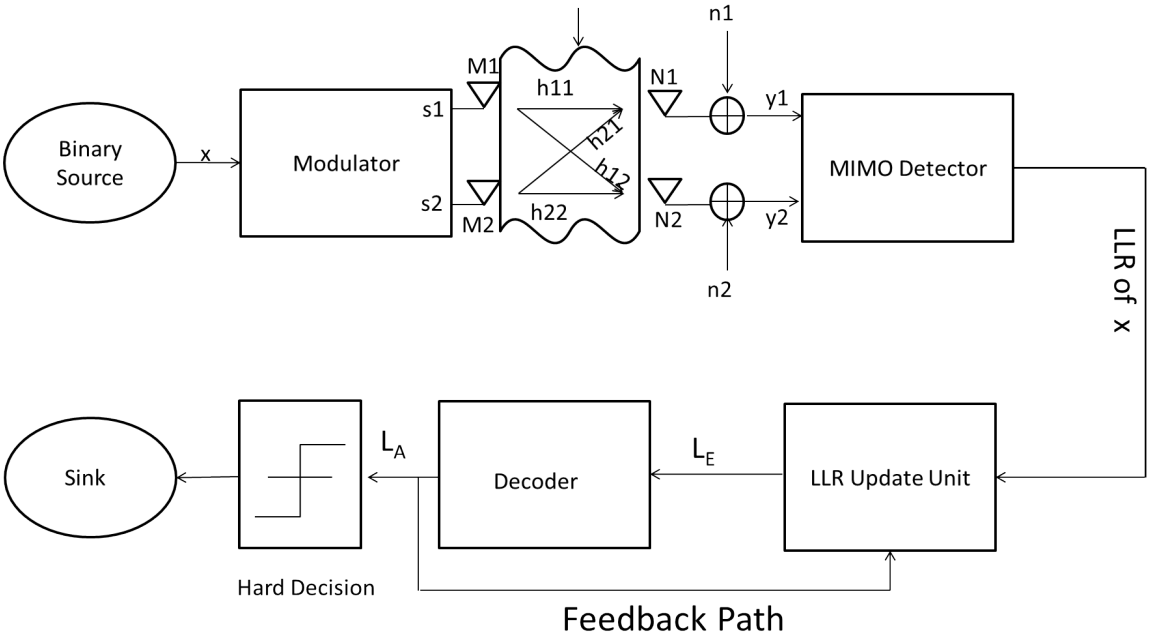

Fig. 22.: Iterative decoding

$$L(x_i) = \ln \frac{P(x_i = 1)}{P(x_i = 0)} \tag{3.1}$$

The a priori LLRs as defined by equation (3.1) are generated by decoder. They are fed to LLR Update Unit which uses them to modify extrinsic LLRs L_E defined by equation (2.1). Detector sends these values to a decoder which in turn further refine these soft values and again generates a priori probabilities. It then again sends them back to detector. This process is called one outer iteration. Decoder uses message passing algorithm in which data is transferred back and forth between check nodes and variables node. It does it iteratively until check node data is stabilized at which point it terminates the iterations. These decoder iterations are called inner iterations. Ideally outer and inner iterations can be continued until all check nodes of the decoder are satisfied. However, in the real hardware system this would result more latency and

variable throughput. While in general more number of outer and inner iterations will improve the error rate performance, interestingly at higher SNRs one can find a trade off between performance and complexity. To implement iterative decoding one needs to generate a candidate list of probable symbol vectors. This list is used to generate LLRs of each bit in the received symbols vector y using max-log approximation. It is shown by equation (3.2)

$$\begin{aligned} L_E(x_i|y) \approx & \frac{1}{2} \max_{x \in X_{i,+1}} \{-\frac{1}{\sigma^2}||y - Hs||^2 + x_{[i]}^T . L_{A,[i]}\} \\ & - \frac{1}{2} \max_{x \in X_{i,-1}} \{-\frac{1}{\sigma^2}||y - Hs||^2 + x_{[i]}^T . L_{A,[i]}\} \end{aligned} \tag{3.2}$$

Where $X_{i,+1}$ and $X_{i,-1}$ are sets of $2^{MQ} - 1$ bit vectors whose i^{th} bit is +1 and -1 respectively. $x_{[i]}$ and $L_{A,[i]}$ are the subvectors of x and LLR removing i^{th} bit respectively. Details regarding derivation of equation (3.2) can be found in [9]. In [15] authors have shown that equation (3.2) can be made more implementation friendly by modifying it according to equation (3.3)

$$\begin{aligned} L_E(x_i|y) \approx & \frac{1}{2} \max_{x \in X_{i,+1}} \{-\frac{1}{\sigma^2}||y - Hs||^2 + x^T . L_A\} \\ & - \frac{1}{2} \max_{x \in X_{i,-1}} \{-\frac{1}{\sigma^2}||y - Hs||^2 + x^T . L_A\} \\ & - L_A(x_i) \end{aligned} \tag{3.3}$$

In the modified version the terms of the max operation need to be computed once per bit vector instead of MQ times. This essentially reduces the amount of computation by half. Following Figure 23 summarizes the process of iterative decoding.

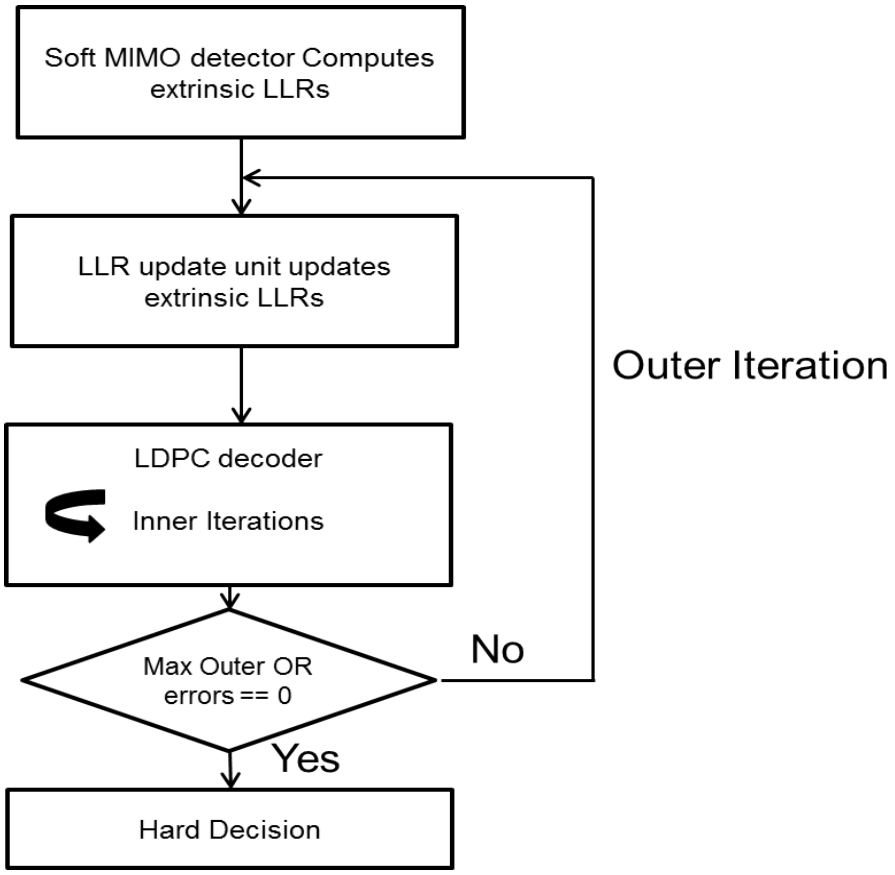

Fig. 23.: Iterative decoding flow chart

1. Simulation Results

LSD is a widely used technique to generate candidate list. In [15] authors have used a LSD along with K-best to show the improving FER performance as one increases number of inner and outer iterations. Hence for simulations we have used LSD algorithm. We have considered a 4x4 MIMO system and a rate 1/2 regular LDPC code of block length 1920 for the simulations. Number of block transmissions done per SNR are 1500. SNR is defined by the equation (3.4).

$$\frac{E_b}{N_0}\Big|_{dB} = \frac{E_s}{N_0}\Big|_{dB} + C \tag{3.4}$$

where C = 0 for QPSK and $C = -10\log_{10} 2$ for 16-QAM, $E_s = E||s||^2$ and $N_0 = 2\sigma^2$ [9].

For Figures 24 and 25 number of inner iterations are kept at 25 and 15 respectively and number of outer iterations are varied from 0 to 4. A 16-QAM modulation scheme is used with a list size of 1024. As one can see in Figure 24 that going from 0 outer iterations to higher outer iterations enhance the BER performance by almost 1.2 dB. However as one further increases the number of outer iterations SNR gain is only marginal ∼0.1dB. Similar results can be seen for FER as shown in Figure 25 where going from 0 to 2 outer iterations dramatically improves the performance by almost 1.5dB.

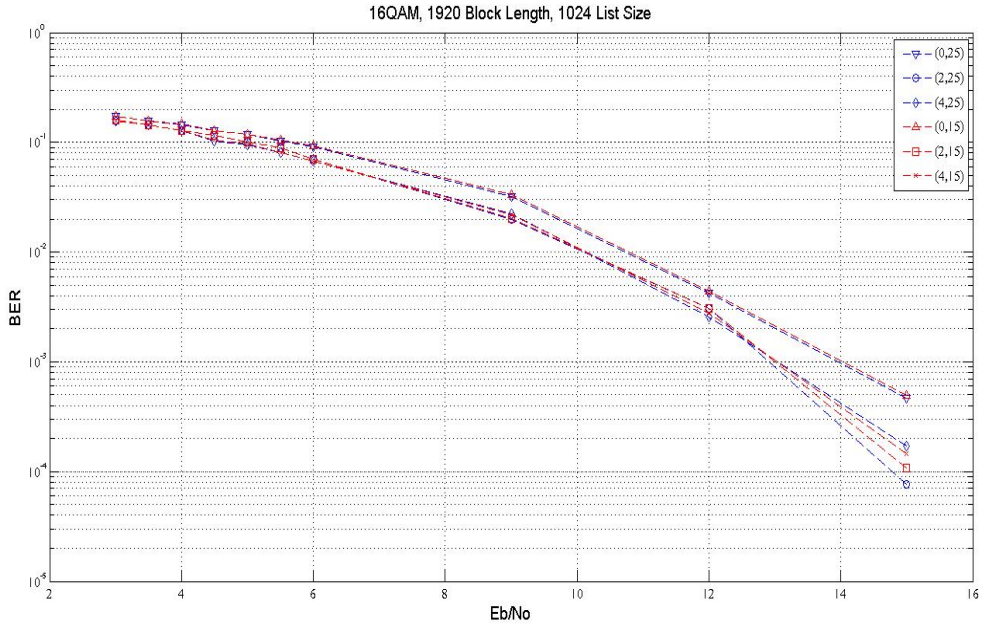

Fig. 24.: BER performance in iterative detection decoding, 16-QAM , 1920 block length, (p,q) refers to p outer and q inner iterations

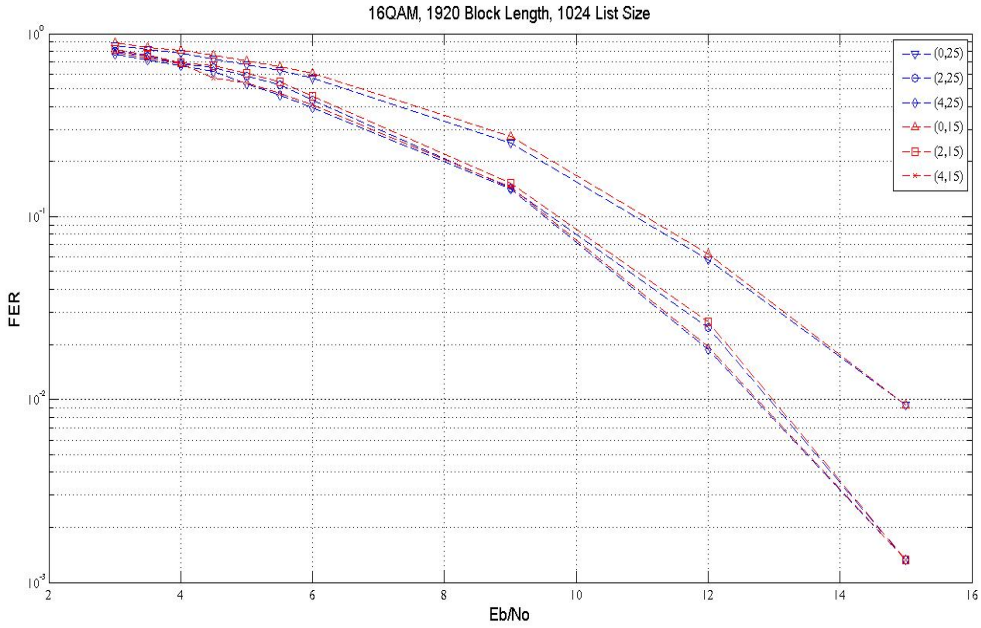

Fig. 25.: FER performance in iterative detection decoding, 16-QAM , 1920 block length, (p,q) refers to p outer and q inner iterations

The effect of varying outer iterations on error rate performance can be seen in Figures 26 and 27 respectively.

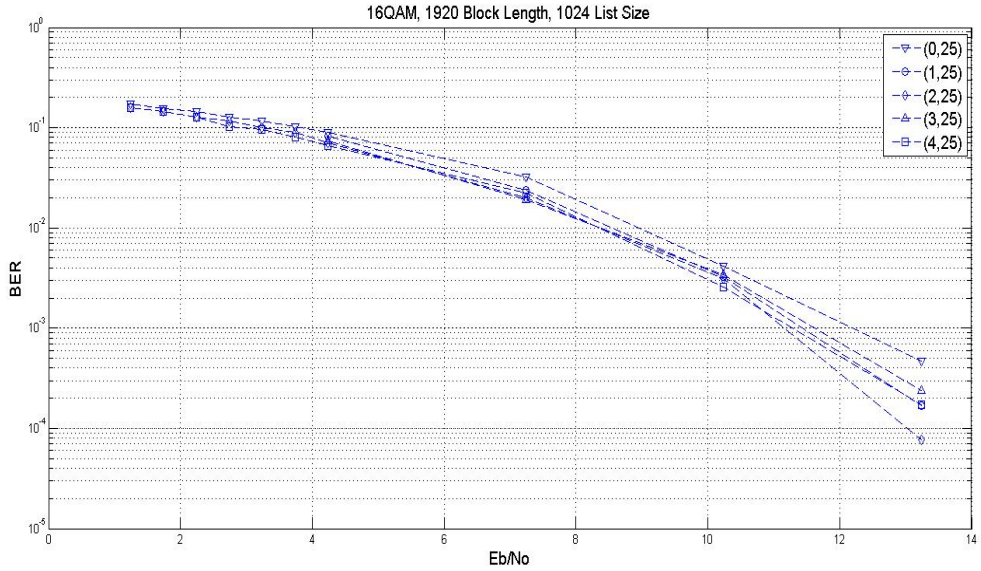

Fig. 26.: BER performance for 25 inner iterations and varying outer iterations from 0 to 4

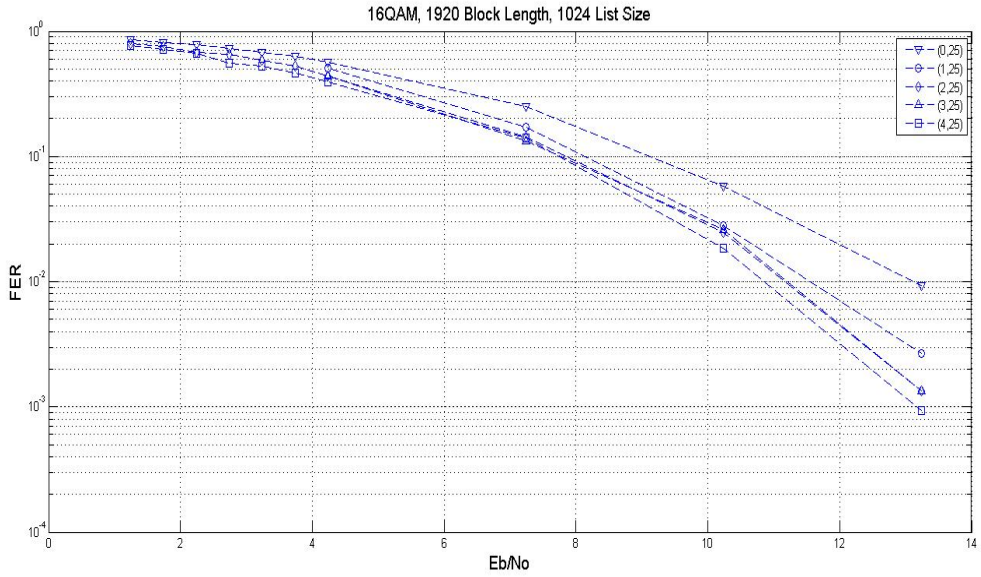

Fig. 27.: FER performance for 25 inner iterations and varying outer iterations from 0 to 4

To measure the effect of list sizes on system's performance we have run simulations with varying list sizes and keeping the number of inner iterations constant at 15. The following Figures 28 and 29 show the performance. A list size of 64 shows worst performance than higher list sizes. As list size increases error rate performance improves. Improvement in performance saturates at higher SNR where varying list size from 512 to 1024 does not give significant performance advantage.

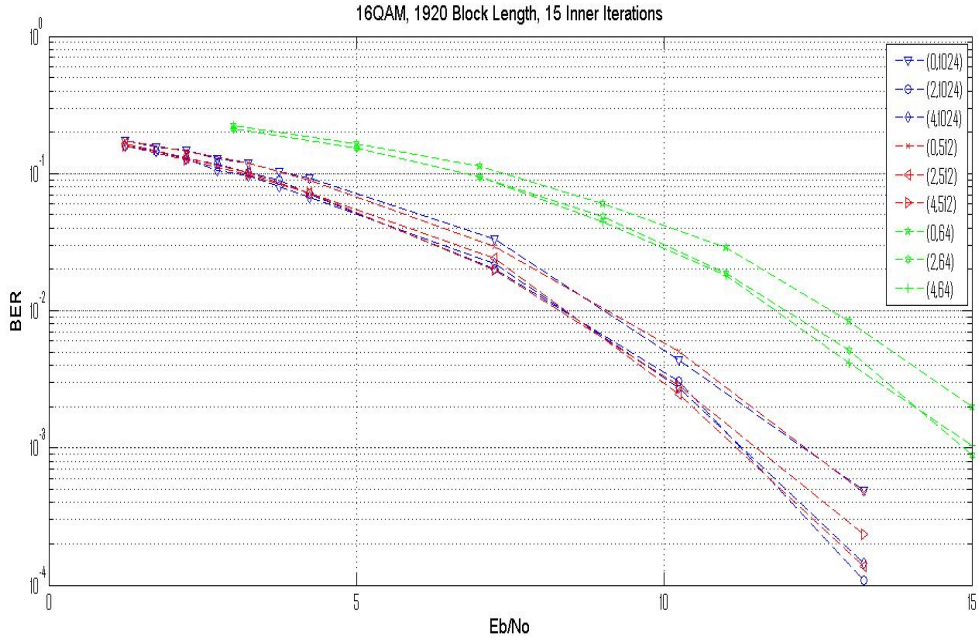

Fig. 28.: BER performance in iterative detection decoding, 16-QAM , 1920 block length, (p,q) refers to p outer iterations and list size of q

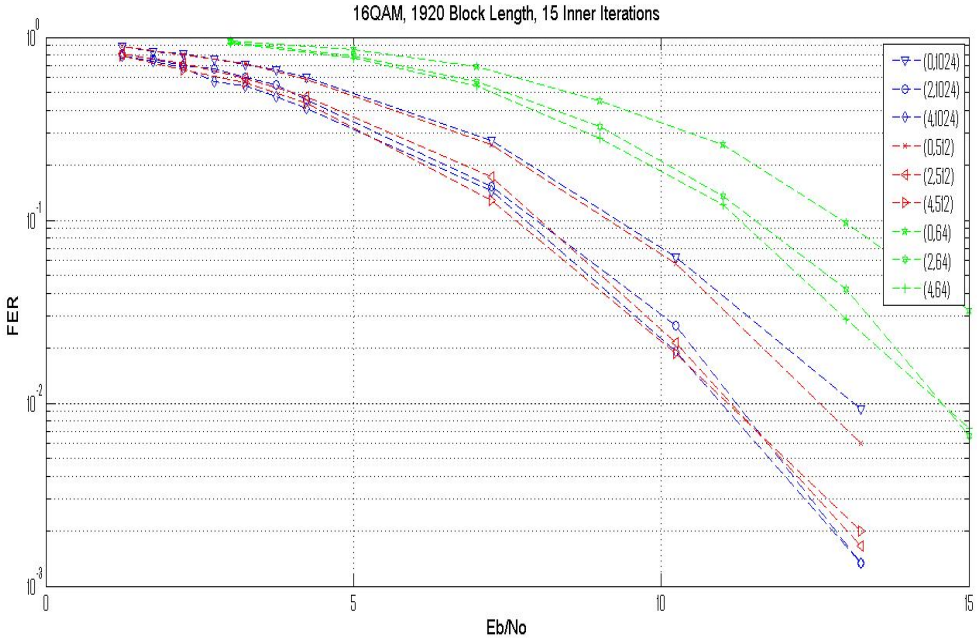

Fig. 29.: FER performance in iterative detection decoding, 16-QAM , 1920 dlock length, (p,q) refers to p outer iterations and list size of q

CHAPTER IV

CONCLUSION AND FUTURE WORK

We have shown the implementation of Soft MIMO detection algorithm on GPUs and compared the performance between two different GPUs and a CPU. We demonstrated that a sphere detector running on a GPU can achieve a throughput of the order of a 4 Mbps which is sufficient for tasks like voice communication and playing online video games. Compare to CPU a throughput increase of $\sim 7x$ is achieved. Comparison of the performances between two GPUs, one with low computational power and one with high computational power shows effect of thread serialization on algorithms with the lower end GPU's execution time curve has a slope of 1/2.

We also did a study on iterative detection and decoding methods and found out that they can be employed to increase MIMO communication system's error rate performance with virtually no change in the existing hardware. We have practically demonstrated that number of inner and outer iterations can be decided based on current SNR, list size and modulation scheme. Hence for a mobile communication system it would mean that at high SNR conditions like when user is close to the baseband station one can accomplish acceptable BER and FER performance with low number of outer and inner iterations. Whereas if the SNR is low one can set higher number of inner and outer iterations or a combination thereof. Iterative decoding technique shows a SNR gain of $\sim 1.5dB$ is achieved when number of outer iterations are increased from zero to two. We showed that where a candidate list of 64 was not sufficient for acceptable error rate performance for a 4x4 MIMO system using 16-QAM modulation scheme, performances were comparable with the list size of 512 and 1024 respectively.

Since we already have a soft MIMO detector on GPU, in the next phase we would like to extend this system and implement decoder and the LLR update unit also on the GPU. It would be an interesting research challenge to run the whole system on a GPU. If we could show that the whole system is getting a desirable throughput then GPU could provide an alternate environment for baseband processing.

REFERENCES

[1] A. Burg, "VLSI Circuits for MIMO Communication Systems," PhD. dissertation, Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology, Zurich, Switzerland, 2006.

[2] "OpenCL - The open standard for parallel programming of heterogeneous systems," Khronos Group, Beaverton, Oregon [Online]. Available: http://www.khronos.org/opencl/

[3] "AMD Accelerated Parallel Processing: OpenCL Programming Guide," AMD Inc, Sunnyvale, CA, 2011.

[4] "The OpenCL Specification," Khronos Group, Beaverton, Oregon [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf

[5] P. Bhagawat, R. Dash, and G. Choi, "Dynamically reconfigurable soft output MIMO detector," in *Proc. of International Conference on Computer Design,* Lake Tahoe, CA, Oct. 2008, pp. 68-73.

[6] R. Wang and G.B. Giannakis, "Approcahing MIMO channel complexity with reduced-complexity soft sphere decoding," in *Proc. of IEEE Wireless Communications and Networking Conf. (WCNC),* vol. 3, March 2004, pp. 1620-1625.

[7] C. Studer, A. Burg and H. Bolcskei, "Soft-output sphere decoding:Algorithms and VLSI implementation," *IEEE J on Sel. Areas Commun.,* vol. 26, no. 2, pp. 290-300, Feb. 2008.

[8] M. S. Khairy, Christian Mehlfuhrer, and Markus Rupp, "Boosting sphere decoding speed through graphic processing units," in *Proc. 16th European Wireless Conference,* Lucca, Italy, April 2010, pp. 99-104.

[9] B. M. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. Commun.,* vol. 51, no. 3, pp. 389-399, Mar. 2003.

[10] J. Boutros, N. Gresset, L. Brunel, and M. Fossorier,"Soft-input soft-output lattice sphere decoder for linear channels," in *Proc. IEEE GLOBECOM,* San Francisco, CA, Dec. 2003, vol. 3, pp. 1583-1587.

[11] H. Vikalo, B. Hassibi, and T. Kailath, "Iterative decoding for MIMO channels via modied sphere decoder," *IEEE Trans. Wireless Commun.,* vol. 3, no. 6, pp. 2299-2311, Nov. 2004.

[12] M. Tuchler, A. C. Singer, and R. Koetter,"Minimum mean squared error equalization using a priori information," *IEEE Trans. Signal Process.,* vol. 50, no. 3, pp. 673-683, Mar. 2002.

[13] B. Steingrimsson, T. Luo, and K. M. Wong,"Soft quasi-maximum-likelihood detection for multiple-antenna wireless channels," *IEEE Trans. Signal Process.,* vol. 51, no. 11, pp. 2710-2719, Nov. 2003.

[14] "MATLAB plug-in for CUDA," NVIDIA Corporation, Santa Clara, CA [Online]. Available:http://developer.nvidia.com/object/matlab_cuda.html

[15] H. Kim, D. Lee, and J. D. Villasenor, "Design tradeoffs and hardware architecture for real-time iterative MIMO detection using sphere decoding and LDPC coding," *IEEE J on Sel. Areas Commun.,* vol. 26, no. 6, pp. 1003-1014, August 2008.

VITA

Richeek Arya received his B. Tech degree in electronics and communication engineering from the International Institute of Information Technology, Hyderabad (IIIT-H), India in 2009. He was involved in challenging projects. He implemented a SQL query processor on a FPGA as a final year B. Tech project. He started his masters in computer engineering in August 2009 and joined Prof. Seong Gwan Choi's group for research. His research interests include high performance architectures for MIMO systems, computer aided design (CAD) and GPU algorithms. While pursuing the masters, he was a teaching assistant for the course ECEN-248 (Introduction to Digital Systems Design). He is also a Texas Public Education Grant holder based on his consistent excellence in academics. During his Masters in the summer of 2010, he did a co-op in Advanced Micro Devices (AMD) in Austin, TX. He was with the Product Development Engineering team in the ATE Characterization group. His responsibilities included creating characterization flow and debugging potential bugs in the first silicon. Impressed by his work, his manger extended to him an offer to continue for the fall 2010 semester. He will join Intel, Santa Clara, CA as a computer aided design engineer after his graduation in August 2011. Richeek Arya can be reached at Office #333G C/O Prof. Gwan Choi, Computer Engineering Group, E.C.E. Department, Wisenbaker Research Center, Texas A&M University, College Station, TX 77843. He can be reached electronically at richeek.arya@gmail.com.

The typist for this thesis was Richeek Arya.