RICE UNIVERSITY

Distributed Partial Decoding in Cooperative Communication Systems

by

Marjan Karkooti

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Dr. Joseph R. Cavallaro, Chair
Professor of Electrical and Computer Engineering

Dr. Behnaam Aazhang
J.S. Abercrombie Professor of Electrical and Computer Engineering

Dr. Mark Embree
Associate Professor of Computational and Applied Mathematics

Dr. Christopher Dick
Adjunct Faculty, Electrical and Computer Engineering
Chief DSP Architect, Xilinx Inc.

Houston, Texas
May, 2009
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.
In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
ABSTRACT

Distributed Partial Decoding in Cooperative Communication Systems

by

Marjan Karkooti

Increasing demand for wireless services is putting major pressure on network resources, which demands a new paradigm with a more efficient design. Recently, cooperative communications has emerged as a viable option for future wireless devices. Major improvements have been made in the theoretical analysis of cooperative communications and relay channels in recent years. But, most of the analyses have some simplifying assumptions that may not be valid in practice. These assumptions include using infinite-length block codes, zero processing delay, etc.

This thesis considers cooperative communications from a practical point of view and identifies the benefits of cooperation when some of the theoretical assumptions are relaxed or changed. Several techniques are introduced to reduce the complexity of the system with minimal performance loss. We set up a framework for system design and show adaptability of our techniques to different scenarios. Our main focus is on the decode-and-forward relaying strategy with low density parity check (LDPC) codes.

The thesis contributions in the field of cooperative communications fall into two main categories: algorithms and architectures. First, we focus on the complexity reduction in the algorithms and propose 'distributed partial decoding'. We demon-
strate the benefits of partially decoding the codeword at the relay and distributing the decoding load between the relay and the destination. This results in major savings in terms of processing power and time at the relay with a very small loss in system performance. The architectural complexity and overhead of this scheme is much smaller than the original decode and forward strategy.

The second contribution of this thesis is the design and implementation of a flexible LDPC decoder architecture that supports a family of LDPC with a variety of code rates and block lengths. This architecture is very suitable for cooperative environments where the cooperating pair and channel conditions and hence the code parameters are not known in advance.

The third contribution relates to leveraging puncturing in cooperation. This work is the first to analyze cooperative communication with punctured LDPC codes. We propose structured puncturing patterns for quasi-cyclic LDPC codes and analyze the tradeoffs in designing good puncturing patterns for cooperative environments.
Acknowledgments

I would like to thank my dear husband, Mahmood, whose love and support has energized me throughout my graduate years. My sincere thanks goes to my thesis advisor Dr. Joseph R. Cavallaro whose guidance and support have been invaluable in this research from the beginning till the end. I would like to thank my committee members Dr. Behnaam Aazhang, Dr. Mark Embree and Dr. Christopher Dick for their insightful suggestions and comments.

I would like to thank all of my friends at Rice and Texas A&M Universities for making my graduate years so memorable. I would like to thank my family: my mother, Maryam Athari, whose unconditional love and wisdom have motivated me to push all the limits, and my brothers who have always supported me emotionally. I would like to dedicate this thesis to the memory of my father, Feridoun Karkooti, who lives in my heart as long as I live.

This work was supported in part by Nokia Corporation, Nokia Siemens Networks, and by NSF under grants CCF-0541363, CNS-0551692, and CNS-0619767.
# Contents

Abstract ........................................... ii
Acknowledgments ................................... iv
List of Illustrations ............................... viii
List of Tables ..................................... xii

1 Introduction .................................... 1
   1.1 Motivation .................................. 1
   1.1.1 Scope of the Thesis ...................... 4
   1.2 Thesis Contributions ....................... 6

2 Background and Related Work ............... 10
   2.1 Low Density Parity Check Codes ............ 11
      2.1.1 Definition of LDPC Codes .............. 12
      2.1.2 Related Work ........................... 13
      2.1.3 Encoding ............................... 19
      2.1.4 Iterative Decoding Algorithms for LDPC Codes 19
      2.1.5 Degree Distribution of LDPC Codes ...... 25
      2.1.6 Gaussian Approximation of Threshold .... 26
   2.2 Cooperative Communications ................. 29
      2.2.1 User Cooperation ....................... 30
      2.2.2 Related Work ......................... 31
   2.3 Puncturing ................................ 35
      2.3.1 Related Work ........................... 35
   2.4 Simulation Environment ...................... 38
2.5 Conclusions .......................................................... 39

3 High Throughput Flexible Architecture for Low Density Parity Check Codes .................................................. 40
  3.1 LDPC Decoder Architecture for Block-Structured Irregular LDPC Codes .................................................. 42
  3.1.1 Scalable Decoder Architecture for Irregular Codes .......................................................... 44
  3.1.2 Hardware Overhead .......................................................... 55
  3.1.3 Hardware Implementation of LDPC Decoder .......................................................... 55
  3.1.4 Decoding Throughput and Performance .......................................................... 57
  3.1.5 Conclusions .......................................................... 59

4 Distributed Decoding in Relay Channels .................................................. 61
  4.1 Cooperative Communications .......................................................... 62
  4.1.1 System Model .......................................................... 63
  4.1.2 Experimental Setup .......................................................... 66
  4.1.3 Timing Diagrams .......................................................... 70
  4.1.4 Simulation Results–Decode, No Encode and Forward, DNEF .......................................................... 74
  4.1.5 Simulation Results–Decode Re-encode and Forward, DEF .......................................................... 79
  4.1.6 Comparison of DEF and DNEF .......................................................... 81
  4.2 Distributed Partial Decoding .......................................................... 83
    4.2.1 Performance of Distributed Partial Decoding .......................................................... 86
    4.2.2 Distributed Partial Decoding for Rayleigh Fading Channels .......................................................... 87
    4.2.3 A Protocol for Choosing the Decoding Load at the Relay and Destination .......................................................... 94
  4.3 Conclusions .......................................................... 96

5 Distributed Decoding with Puncturing .................................................. 98
  5.1 Puncturing .......................................................... 98
5.1.1 Puncturing Ratios ........................................ 101
5.1.2 Puncturing Patterns ....................................... 104
5.2 A Metric for Comparing Puncturing Patterns .............. 109
  5.2.1 Examples .................................................. 110
  5.2.2 Metrics ................................................... 120
5.3 Cooperative Communications with Puncturing at the Relay .. 124
  5.3.1 Effect of Decoding Iterations on Puncturing .......... 125
  5.3.2 Distributed Decoding with Puncturing ................. 128
  5.3.3 Other Block Lengths and Code Rates ................. 131
5.4 Incremental Redundancy ....................................... 134
  5.4.1 Hardware Overhead .................................... 138
  5.4.2 Discussion ............................................. 140
5.5 Conclusions .................................................. 141

6 Conclusions and Future Work .................................. 142
  6.1 Conclusions ............................................... 142
  6.2 Future Work .............................................. 145

Bibliography .................................................... 147
Illustrations

1.1 Demonstration of cooperative wireless transmission. 2

2.1 Tanner graph of a parity check matrix. 14
2.2 The $\phi(x) = -\log(\tanh(x/2))$ function. 22
2.3 Demonstration of cooperative wireless transmission. 30

3.1 An overall view of a block-structured irregular parity-check matrix, $N = 1296$ bits and $R = 2/3$. The black squares show a shifted identity matrix and the white squares show a zero matrix. 43
3.2 LDPC decoder block diagram. 45
3.3 The decoding functional unit block diagram. 47
3.4 The serial Min-Sum unit block diagram. 48
3.5 The flexible permuter block diagram. 49
3.6 The check memory organization block diagram. 51
3.7 The variable memory organization block diagram. 52
3.8 FER for irregular block-structured PCM (R=2/3, N=1296, MaxIter=15, layered belief propagation). 58
3.9 Average decoding throughput for different code rates and codeword lengths for the flexible irregular architectures synthesized for FPGA and ASIC based on the average number of iterations. 59
4.1 Half-duplex relay channel. In the first half of the time slot, the source broadcasts its information. In the second half, both source and relay send some information to the destination. .......................... 62
4.2 System model. ............................................. 64
4.3 Source - Relay - Destination Block Diagram. ......................... 67
4.4 Decode, encode and forward (DEF) strategy vs. decode, no encode and forward (DNEF) strategy. ................................. 69
4.5 Timing diagram of Decode and Forward Relaying. Notice that the relay has limited time to decode/re-encode and transmit the received signal. ............................................. 71
4.6 An alternative timing diagram for Decode and Forward Relaying - More processing time at the relay. ................................. 72
4.7 Bit error rate vs. signal to noise ratio for different locations of a relay with respect to the source, Block length=1296 bits, code rate=1/2, $\alpha = 2$, DNEF. ................................. 74
4.8 Effect of power distribution on the performance:(1296,1/2) LDPC code, $\alpha = 2$, DNEF ............................................. 76
4.9 Bit error rate vs. signal to noise ratio. Block length = 1296 bits, effect of code rate on the performance of the direct link. .......... 77
4.10 Effect of code rate on the relay performance. (1296,1/2), $\alpha = 2$, DNEF. 78
4.11 Bit error rate vs. signal to noise ratio. Similar codes for the SR, RD links, Relay: Decode, re-encode and forward ........................ 79
4.12 Different code rates at the source and the relay with equal number of information bits. Source: (1944,1/2), Relay:(1296,3/4). Relay: decode, re-encode and forward, Equal power. .................. 80
4.13 Comparison of the results for DEF and DNEF, Equal power. ....... 82
4.14 Percentage of bits in error vs. Number of decoding iterations for
different SNR values, for an LDPC code of (1296,1/2). Results are for
the direct link case. .................................................. 84

4.15 Distributed and partial decoding, effect of number of iterations at the
relay and the destination on the performance, (1296,1/2) LDPC code. 85

4.16 Distributed and partial decoding, effect of number of iterations at the
relay and the destination on the performance, (1296,1/2) LDPC code. 88

4.17 Distributed and partial decoding, Rayleigh fading channel, (1296,1/2)
LDPC code. .......................................................... 89

4.18 Zoomed version of figure 4.17. .................................. 89

4.19 Percentage of change in the mean of LLR messages from variable
nodes, for a noise level close to the threshold. .......................... 92

4.20 Probability of error vs. iteration number for a noise level close to the
threshold. .............................................................. 94

4.21 Change in the probability of error vs. iteration number for a noise
level close to the threshold. ........................................... 95

4.22 Percentage of change in the probability of error vs. iteration number
for a noise level close to the threshold. .............................. 95

5.1 Block diagram showing puncturing at the relay. ...................... 100

5.2 Puncturing 0%, 25%, 50%, 75% of the bits and sending the rest of the
bits. ................................................................. 101

5.3 Structured puncturing patterns ..................................... 105

5.4 Effect of utilizing structured puncturing patterns at the relay on the
performance of a cooperative communication system. .................. 108

5.5 Puncturing percentage vs threshold for the LDPC code in Ex1 ... 112

5.6 Puncturing percentage vs threshold for the LDPC code in Ex2 ... 116

5.7 Puncturing percentage vs threshold for the LDPC code in Ex3 ... 119
5.8 Cooperative communications with puncturing at the relay, effect of decoding iterations on performance. 126
5.9 Cooperative communications with puncturing at the relay, effect of decoding iterations at the destination on performance. 127
5.10 Effect of distributed and partial decoding with puncturing on the performance. 130
5.11 Distributed and partial decoding, effect of puncturing on the performance. 133
5.12 Block diagram of incremental redundancy. 134
5.13 Graphical representation of the errors in a codeword after each iteration of LDPC decoding. 135
5.14 Histogram of number of bit-errors per each LLR value from a LDPC codeword. 136
5.15 Performance of direct path communication with incremental redundancy. 138
5.16 Frame error rate, performance of direct path communication with incremental redundancy. 139
6.1 Three node cooperative communications with LDPC coding. 144
6.2 Summary of the thesis contributions on a three node cooperative communications block diagram. 145
Tables

3.1 Design statistics for the flexible irregular LDPC decoder on Virtex4-xc4vf60 FPGA. .............................................. 56
3.2 ASIC design statistics for the Dec8 flexible LDPC decoder. ........... 57

5.1 Effective rate and the puncturing percentage of the puncturing patterns.103
5.2 Threshold values for different puncturing experiments for the LDPC code in Ex1, $\lambda(x) = 0.05263x + 0.15789x^5 + 0.26316x^9 + 0.52632x^{19}$, $\rho(x) = x^{18}$ ......................................................... 112
5.3 Threshold values for different puncturing experiments for the LDPC code in Urbanke’s paper ......................................................... 114
5.4 Threshold values for different puncturing experiments for the LDPC code in Ex2 ......................................................... 117
5.5 Threshold values for different puncturing experiments with the LDPC code in Ex3 ......................................................... 119
5.6 Number and percentage of punctured branches for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2 .......... 120
5.7 Average row and column degrees for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2 ............................... 121
5.8 Number of short cycles of length four and six for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2 ............... 123
5.9 The threshold values for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2, assuming infinite length code with the same degree distribution ........................................ 124

5.10 Number and percentage of punctured branches for different puncturing patterns for an LDPC code of length 1944 bits, rate 1/2 . 132

5.11 Average row and column degrees for different puncturing patterns in an LDPC code of length 1944 bits, rate 1/2 ................. 132
Chapter 1

Introduction

1.1 Motivation

The market for embedded devices is growing at an astonishing rate. There is an increasing demand for wireless communications in laptops, PDAs, digital cameras, etc. These devices demand higher data rates, more reliable communications with lower cost and increased battery life. Each year even more of these wireless devices enter the market, all of which are competing for limited shared resources. This puts major pressure on the wireless medium and may result in a shift of paradigm. A similar problem was faced in the 1980s, which led to the shift from analog to digital communications. Cooperative communications is a very promising candidate for being the next new paradigm.

In the current state of the art of wireless systems, single mobile devices interact with a base station. Most of these devices have underutilized radios since for a large fraction of time, they operate in idle mode. This results in inefficient use of hardware resources such as transmit radios.

On the other hand, wireless channels impose noise and attenuations on transmitted signals depending on the physical medium a signal is passing through. Transmit
Figure 1.1: Demonstration of cooperative wireless transmission.

Diversity can be achieved by transmitting multiple independent versions of the signal, which helps to remove the effect of fading. In a mobile environment, the transmitter terminals have limitations in terms of size and power, which limits the number of antennas that they can support. As a result, these units may benefit from cooperation by sharing their antennas.

In cooperative communications, single-antenna mobile nodes form a group with their neighboring nodes and build a virtual multiple-antenna or multiple input multiple output (MIMO) system. These distributed transmitters interact with each other to jointly transmit their information. Therefore, they enjoy transmit diversity, which results in a more robust system. For example, if the link between a node and its serving base station goes into a deep fade, other nodes can help transmit its data.
Figure 1.1 illustrates a mobile network utilizing cooperative wireless communications. In this figure, there are three mobile nodes that communicate with a base station. Each node can send some information to the base station directly (i.e. direct-line transmission) or can help other nodes by relaying their data to the destination. In cooperative communications, mobile nodes cooperate with each other in data transmission. Therefore, the destination node receives multiple versions of the transmitted signal and can detect and decode the signal with higher probability of success because of diversity.

Several issues need to be addressed in a cooperative environment, such as relay selection [1], loss of rate, power [2] and time at the relay due to cooperation. Another important issue is the extra overhead in terms of hardware complexity in each mobile terminal in order to support cooperation.

Information theoretic analysis of cooperation in recent years has shown many promising aspects of this technique. But, many of the papers do not consider the problems faced when developing a practical user cooperation environment such as hardware overhead and complexity. On the other hand, they make some unrealistic assumptions about the relays, without which the mathematical description of the problem could become very difficult or intractable. For example, complete decoding [3] while having zero processing delay at the relay is typically assumed in the analysis.

In this thesis, we consider the problem of cooperation from a practical point of view. This thesis tries to close the gap between mathematical analysis using paper and
pencil and practical problems such as building a system based on the models developed by mathematical analysis. This is an important problem if we want to develop the next generation of wireless systems and make them capable of cooperation. We setup a framework for system design and present techniques for designing a cooperative communication system with reduced complexity both in terms of the algorithm and hardware. Our focus is not only on reducing the processing and retransmission time and power consumption, but also on minimizing the architectural overhead. We propose ways to minimize the processing delay while taking advantage of the relays. Also, a few algorithms are offered to reduce power consumption at the relay while processing the data of other users. In addition, we design hardware that supports cooperation.

1.1.1 Scope of the Thesis

The thesis explores the design space for the practical use of distributed decoding. It sets up a framework for designing a system capable of cooperation. By careful consideration of algorithms and by imposing practical limitations on them, we show how to take advantage of a few well-analyzed approximations to reduce the complexity of the system in terms of hardware architecture and provide great savings in not only processing power and processing time but also in transmission power and time.

There exist a number of different types for cooperation in the literature such as amplify and forward, estimate and forward, and decode and forward. In this thesis,
we focus on decode and forward relaying because it is the most promising scheme in terms of performance while requiring the largest amount of resources at the relay. Since error correction is performed at the relay by utilizing an error correcting code such as LDPC or turbo codes, decode and forward relaying scheme presents the highest gain in relaying. We show how to reduce complexity of a cooperative system as a whole by reducing the complexity of each smaller section while maintaining the relaying gain.

We have chosen the Low Density Parity Check (LDPC) codes as the channel coding for our analysis. LDPC codes show superior performance compared to other types of channel coding, such as turbo codes or convolutional codes. LDPC codes have been adopted in a few standards such as DVB-S2 and are optional forward error correcting codes for IEEE 802.11n [4] and IEEE 802.16e [5] standards. Although LDPC codes offer many benefits, because of many degrees of freedom in their design, there are many challenges in finding an optimal code and designing suitable decoding hardware. These challenges are one of the focuses of this thesis.

Puncturing—or selecting and not transmitting some of the bits—has shown to be beneficial in increasing the spectral efficiency of a system with some known performance loss. To the best of our knowledge, this thesis is the first to analyze cooperative cooperation using punctured LDPC codes and the challenges in designing LDPC puncturing patterns suitable for cooperation. In addition, we propose a new scheme for increasing the performance of a system using puncturing with incremental redun-
dancy. Incremental redundancy refers to the case of transmitting the punctured bits in the later time steps after a failed decoding.

1.2 Thesis Contributions

Contributions of this thesis are threefold, each of which sheds some light on different aspects of a cooperative communications system design framework in terms of algorithmic or architectural design, having in mind the limitations that are faced in practice.

The first contribution of the thesis is proposing Distributed and Partial Decoding [6, 7]in decode and forward relaying. This scheme has many benefits over the classic decode and forward relaying and results in significant savings in terms of the processing power and the processing time at the relay. Through mathematical analysis and extensive simulations we have shown all the benefits of this scheme over the classic decode and forward relaying, with very negligible performance loss.

We have also analyzed the decode and forward relaying (DEF) and proposed decode and forward without re-encoding (DNEF). The benefits of DNEF over the original decode and forward relaying such as smaller area and processing time are shown. Different cases have been considered for comparison from varying code rate to varying power of transmission for the source and the relay. In all of these comparisons we have considered the practical assumptions of finite-length LDPC codes and delay-limited systems.
Considering the fact that in mobile environments, the topology of the system is not known in advance, and each node may act as a relay for a number of nodes at different times, the relay is required to have a flexible architecture capable of adjusting to the needs of different users based on topology, channel conditions, bit error rate (BER) requirements and the tolerable delay for each user. This brings up the need for a flexible architecture at the relay.

Our second contribution is the design and implementation of a flexible LDPC decoder architecture that supports a family of quasi-cyclic structured LDPC codes consisting of three block lengths and four code rates based on the IEEE 802.11n standard [8–10]. This decoder architecture supports three block lengths of 648, 1296, 1944 bits and code rates 1/2, 2/3, 3/4, 5/6. The designed architecture achieves an average throughput of hundreds of Mbps depending on code rate and block length of the supporting LDPC code. The above architecture can be used both as a stand-alone decoder or in a larger system as a node in cooperative communications. The flexibility of this architecture to support a family of LDPC codes makes it very suitable for use in relays. This enables the relays to use one decoder hardware to support different source nodes with different LDPC code requirements.

The third contribution of the thesis is the trade-off analysis of puncturing and incremental redundancy in cooperative communications using LDPC codes. We propose structured puncturing patterns for quasi-cyclic LDPC codes with applications in user cooperation and find the best puncturing patterns for these codes for low and
high signal to noise ratios (SNR). We also show the robustness of the distributed decoding and relay channels to puncturing and show that the codeword at the relay can be punctured with high puncturing ratios and still be beneficial for the destination.

Incremental redundancy refers to the case that the transmitter does not transmit all the parity or redundant bits in one packet with the information bits. If the receiver is unable to decode the first packet, it requests for more bits, and a second set of information is transmitted from the transmitter. All the papers in the literature assume that the receiver starts the second LDPC decoding procedure from scratch. We show that there is useful information hidden in the results of the first decoding procedure. By leveraging this information, we show that we can achieve savings in terms of processing time and power.

The outline for this thesis is as follows. In Chapter 2, we present background and related work in the literature in decode and forward schemes for relays. We review the recent results in this area and highlight areas for improvement which have led to the contributions presented in Chapters 3 through 5. Since this thesis has three major sections for architectures and algorithms, we present the background for all three sections, LDPC decoding, cooperative communications and puncturing in the second chapter.

Chapter 3 introduces the architecture that we have designed for LDPC codes that is flexible to support a variety of codes in the family of quasi-cyclic codes. This flexible decoder will be used in the remainder of the thesis.
In Chapter 4, a description of our contributions in reducing the complexity of cooperative communications with decode and forward relaying is presented. We show how to reduce the processing time and processing power at the relay by distributing the decoding load between the relay and the destination.

In Chapter 5, cooperative communication utilizing punctured LDPC codes is presented. We evaluate the trade-offs of designing good puncturing patterns for cooperative environments using LDPC codes. Furthermore, some techniques for reducing the transmission time/power at the relay through puncturing are offered. Incremental redundancy is also considered for cooperative environments with feedback. We show how to reduce the complexity of these systems by leveraging the information hidden in the decoders.

Chapter 6 concludes the thesis. In addition, a few ideas for extension of the current work are presented in this chapter.
Chapter 2

Background and Related Work

In wireless communications several measures are used to achieve error-free communications, one of which is using error correcting codes such as Viterbi [11, 12], turbo [13–15] or LDPC codes [16–18] in channel coding. All of these codes have error detection and correction abilities. Error detection is the ability to detect the presence of errors caused by noise or other impairments during transmission. Error correction is the ability to reconstruct the original data by taking advantage of the redundant data.

There are two basic protocols for designing a system: Forward Error Correction (FEC) and Automatic Repeat Request (ARQ) [19]. In forward error correction, the receiver encodes the data with an error correcting code and transmits it to the receiver. The receiver decodes the data and tries to detect and correct as many errors as possible. By using strong error correcting codes, the receiver is able to decode the data correctly with high probability. A system using FEC codes usually does not require any feedback. In most of the thesis, a system with forward error correction is utilized.

In automatic repeat request, the transmitter sends the codeword that is encoded with an error detection code. The receiver checks for errors and in case of detecting
an error, it requests for retransmission. In many cases, this request is implicit. If the transmitter does not receive an acknowledgement, it will retransmit the data.

The combination of the above techniques is called hybrid automatic repeat request or HARQ. In HARQ, minor errors can be corrected without retransmission and major errors are detected and retransmission is requested. We will use this technique while talking about puncturing and incremental redundancy in chapter 5.

The organization of this chapter is as follows. In section sec:LDPC, an overview of LDPC codes and their decoding algorithm is presented. Section 4.1 talks about cooperative communications using LDPC codes. Last section 2.3 gives an overview about puncturing.

2.1 Low Density Parity Check Codes

Low density parity check codes are one of the best error correcting codes [20] that are available today. LDPC codes were proposed by Gallager [16] more than 40 years ago, but his work received little attention until after the invention of turbo codes [13], which used the same concept of iterative decoding. In 1996, MacKay and Neal [17,21,22] rediscovered LDPC codes. The excellent error correction capability of these codes led to an overwhelming interest in designing LDPC codes suitable for practical applications.

One of the major applications for error correcting codes such as convolutional code, turbo or LDPC codes is wireless communications. Every year millions of wireless
devices enter the market. There is an increasing need for higher data rate and higher
throughput, which requires the use of stronger and more powerful error correcting
codes. The LDPC and turbo codes have been shown to be the two major competitors
in this space. Many standards have adopted or are considering LDPC codes, such
as DVB-S2, IEEE 802.11n [4] and WiMAX. Other standards, such as DVB-S and
3GPP-LTE, use turbo codes.

2.1.1 Definition of LDPC Codes.

Low Density Parity Check codes are a class of linear block codes specified by a very
sparse parity check matrix (PCM) $H_{(N-K)\times N}$. Each element of the PCM is either
a zero or a one, where nonzero entries are typically placed at random. During the
encoding process, $N - K$ redundant bits are added to the $K$ information bits to
form a codeword length of $N$ bits. The code rate is the ratio of the information bits
to the total bits transmitted in a codeword ($R = K/N$). LDPC codes are usually
represented by a bi-partite graph called a ‘Tanner graph’. Figure 2.1 shows a Tanner
graph [23], [24] of a simple parity check matrix $H_{4\times 8}$. There are two classes of nodes in
a Tanner graph, ‘Variable’ nodes and ‘Check’ nodes. A variable node corresponds to a
‘coded bit’ or a PCM column, and a check node corresponds to a parity check equation
or a row of the PCM. In this graph, variable and check nodes have degrees of two
and four, respectively. There is an edge between each pair of nodes if there is a ‘one’
in the corresponding PCM entry. During the decoding process, messages are passed
among the graph nodes. Log-likelihood ratios (LLRs) are used for representation of reliability messages. Ryan [25] has a very good tutorial on LDPC codes; some of the descriptions in this work has been taken from his document.

The number of nonzero elements in each row or column of a PCM is called the 'degree' of that node. A cycle of length $l$ in a Tanner graph is a path comprised of $l$ edges which closes back on itself. The Tanner graph in the above figure has a cycle of length four, which has been shown by dashed lines. Girth of a Tanner graph is the minimum length of the cycles in the graph. The shortest possible cycle in a bipartite graph is clearly a length-4 cycle. Length-four cycles manifest themselves in the $H$ matrix as four 1's that lie on the corners of a sub-matrix of $H$.

An LDPC code is regular or irregular based on the node degrees. If variable or check nodes have different degrees, then the LDPC code is called 'irregular' otherwise, it is called 'regular'. In low SNR regimes, irregular codes usually have better performance than regular codes [26] because the nodes with higher degrees converge faster and assist the nodes with lower degrees. On the other hand, the irregularity of the code results in a more complex hardware architecture.

2.1.2 Related Work

In the last few years extensive research has been done on designing architectures for LDPC coding. Researchers have been looking for the best trade-off between area, data rate and power consumption. In this section, we present some of the novel
ideas in this field. [27] designs LDPC codes for binary erasure channels (BEC) with less number of decoding iterations in the parallel message-passing decoding. They consider the cases where the capacity is approached with sufficiently low parity-check matrix density. [28] analyzes LDPC code families that support variety of different rates while maintaining the same fundamental decoder architecture. They combine rows of the lowest-rate PCM to produce the PCMs for higher rates with the same block length.

Designing hardware architectures for LDPC codes falls into three different categories. Fully parallel, semi-parallel and serial. Fully parallel architectures have as many processing units as the number of nodes in the Tanner graph [23] of a code. In Fully parallel architectures, all the functional units are connected with wires according
to the Tanner graph connections. Authors in [29] directly mapped the Sum-Product decoding algorithm to hardware employing a fully parallel approach. Although their design has very good performance, the large area and the routing complexity and overhead makes this approach infeasible for larger block lengths (e.g., more than 1000 bits). One of the drawbacks of a fully parallel architecture is its lack of flexibility in the decoder architecture.

In a semi-parallel approach, some of the processing units are reused to decrease the area at the expense of lower throughput comparing to fully parallel. In [30], an FPGA implementation of a (3, 6) regular LDPC semi-parallel decoder is offered. In this design, a multi-layered interconnection network is used to access messages from memory. The authors in [31] proposed a low-power 1055 bit, rate 0.4, (3, 5) regular semi-parallel decoder architecture. A fully structured parity check matrix was used to eliminate the data dependence among the neighboring nodes, which led to a simpler memory addressing scheme than [30]. One important family of the structured LDPC codes is the Quasi-cyclic codes in which a parity check matrix is made of several sub-blocks each of which are shifted version of an identity matrix of size $S \times S$. By using a semi-parallel approach, several processing units – equal to the size of the sub-blocks $S$ – can work simultaneously to reduce the decoding time and hence increase the throughput by a factor of $S$ compared to the serial approach.

Most of the papers related to architectures for LDPC decoders are based on a fixed code [32, 33] or they are scalable and support multiple rates, but only for a
single code size. For example, a scalable structured LDPC decoder with relatively high data throughput is proposed in [34] for very long codeword lengths defined by the DVB-S2 standard. Also, authors in [35], [36] and [37] offered multi-rate decoder designs based on structured parity check matrices (PCMs) for regular and irregular codes, using the standard belief propagation algorithm.

[31] designed an LDPC decoder architectures based on the layered belief propagation (LBP) algorithm. In recent years, several other architectures have been proposed for decoding LDPC codes. Most of them use Quasi-cyclic LDPC codes with variations of the sum-product decoding algorithm. To name a few, the reader is referred to [38–40].

In chapter 3, we present design of a flexible decoder architecture for LDPC codes [8, 9]. There are some recent papers on LDPC architecture that have built upon our work such as [37, 41–44].

Now, we refer to some more recent papers in the area of architecture design for LDPC coding. An efficient highly-parallel decoder architecture using partially overlapped decoding scheme for quasi-cyclic (QC)-LDPC codes is proposed in [45]. For a (3, 5)-regular QC-LDPC code, they report reducing the hardware complexity by approximately 33% for the CNU and 20% for the VNU in the highly-parallel decoder architecture without any performance degradation.

[46] presents a high-throughput decoder architecture for QC-LDPC codes. They propose a row permutation scheme to significantly simplify the implementation of the
shuffle network in the decoder. It is estimated that their architecture can support 4.7 Gb/s decoding throughput with 15 decoding iterations.

An algorithm that can efficiently generate all the control signals for the shuffle network used in flexible LDPC decoders is proposed in [47]. This algorithm reduces the hardware complexity of the controller of shuffle networks using the Benes network structure. The authors propose an efficient shuffle network for WiMAX LDPC decoders.

Authors in [48] present construction of a new class of implementation-oriented LDPC codes, namely shift-LDPC codes. With girth optimization, Shift-LDPC codes perform as well as computer generated random codes with very high decoding speeds. They present an ASIC design utilizing 0.18μm CMOS technology for a 8192-bit regular LDPC code which can achieve 5 Gb/s throughput at 15 iterations.

Low-density parity-check convolutional codes (LDPC-CCs) are investigated in [49]. LDPC-CC codes demonstrate comparable error-correcting performance to LDPC block codes (LDPC-BCs). However, the LDPC-CC encoder requires termination when applied to finite-length data frames to ensure that the trailing information bits are fully protected. The authors proposed a high-speed architecture for LDPC-CC encoders with built-in termination. Synthesis results for LDPC-CCs of code memory size up to 512 demonstrate maximum encoding throughputs of around 1 Gb/s for a 90nm CMOS technology.

A base-matrix based decoder architecture for multi-rate QC-LDPC codes for
broadband broadcasting systems is presented in [50]. The authors designed a decoder utilizing Modified Min-Sum Algorithm (MMSA) that is scalable in throughput and flexible in code rate and code length.

[52] presents high-throughput design approaches for QC-LDPC decoders. They implemented QC-LDPC decoders for Chinese Digital Television Terrestrial Broadcasting using field programmable gate array (FPGA). They propose methods to improve the throughput performance, as well as the throughput-and-hardware tradeoff, of decoders with semi-parallel architecture.

An efficient low-complexity switch network for reconfigurable LDPC decoders is proposed in [53]. In addition, they propose an algorithm to generate all the control signals for structured QC-LDPC decoders. In synthesis result using the TSMC 0.18-μm standard cell CMOS technology, the proposed switch network for a reconfigurable LDPC decoder of IEEE 802.16e and IEEE 802.11n can be implemented with an area of 0.772mm², which leads to a significant area reduction.

A low-complexity reconfigurable VLSI architecture for high-speed LDPC decoders is presented in [54]. Shift-LDPC codes are incorporated within the design, and have shown not only comparable decoding performance to computer generated random codes but also high hardware efficiency in high-speed applications. The ASIC implementation results of an (8192, 7168) (4, 32)-regular shift-LDPC decoder demonstrate a maximum decoding throughput of 3.6Gb/s at 16 iterations.
2.1.3 Encoding

To encode a message $m$ of $K$ bits with LDPC codes, $C_{1 \times N} = M_{1 \times K} \cdot G_{K \times N}$ should be calculated, in which $C$ is the codeword and $G$ is the generator matrix of the code which is the null matrix of the parity check matrix $H$ ($GH^T = 0$). All the computations in the encoding procedure are on binary values and in bit-level that can be done efficiently using XORs and AND gates.

Several low complexity algorithms exist for encoding of LDPC codes. Some techniques exploit the sparseness of the parity check matrix for efficient encoding [30]. Others impose some structure to the Tanner graph so that encoding is transparent and simple. Repeat-Accumulate codes are an example of structured graphs. Richardson et al. proposed that transforming the generator matrix to upper triangular form leads to reduced complexity encoding [55].

2.1.4 Iterative Decoding Algorithms for LDPC Codes

Gallager presented a decoding algorithm that is effectively optimal in his work in 1960. Since then, other researchers have independently discovered that algorithm and related algorithms, although sometimes for different applications. Several names exist for this algorithm, such as: 'Message Passing', 'Sum-Product (SP)' or 'Belief Propagation (BP)'. It iteratively computes the probability distributions of the variables in graph-based models. The iterative decoding algorithm for turbo codes is a specific instance of the Sum-Product algorithm.
During the decoding process of LDPC codes, information is sent along the edges of the Tanner graph (see the graph in Fig. 2.1). Local computations are done in each node of the graph. In the rest of this section, we will describe the Sum-Product algorithm and some of its variations such as the Modified Min-Sum (MMS) algorithm and the layered belief propagation algorithm.

Throughout this thesis document, an AWGN (Additive White Gaussian Noise) channel is considered with BPSK (Binary Phase Shift Keying) modulation of the signals unless otherwise stated.

**Sum-Product Algorithm**

Sum-product algorithm is an iterative algorithm that corrects the errors in a LDPC code iteratively. Let $R_{mj}$ denote the check node log likelihood ratio (LLR) message sent from the check node $m$ to the variable node $j$. Let $L(q_{mj})$ denote the variable node LLR message sent from the variable node $j$ to the check node $m$. The $L(q_j)$ ($j = 1, \ldots, N$) represent the *a posteriori* probability ratio (APP messages) for all variable nodes (coded bits). The APP messages are initialized with the corresponding *a priori* (channel) reliability value of the coded bit $j$. In each iteration, some of the APP messages corresponding to the code bits that are received in error are corrected. Theoretically, the received codeword converges to the transmitted codeword after several iterations.

The Sum-Product algorithm operates on the nonzero entries of the parity check
matrix $H$. It iterates over the columns and rows of the PCM performing the following steps:

Step 1) Initialization: For each variable node $j$, messages $L(q_{mj})$ that correspond to a particular check equation $m$ are computed according to:

$$L(q_{mj}) = L(q_j) = \frac{2y_j}{\sigma^2},$$

in which $y_j$ is the received signal and $\sigma$ is the channel noise.

Step 2) For each check node $m$, messages $R_{mj}$ corresponding to all variable nodes $j$ that participate in a particular parity-check equation are computed according to:

$$R_{mj} = \prod_{j \in N(m)\setminus\{j\}} \text{sign}(L(q_{mj})) \phi \left[ \sum_{j \in N(m)\setminus\{j\}} \phi(L(q_{mj})) \right],$$

where $N(m)$ is the set of all variable nodes from parity-check equation $m$. $\phi(x)$ can be represented with the following equation:

$$\phi(x) = -\log(\tanh(x/2)) = \log\left(\frac{e^x + 1}{e^x - 1}\right).$$

The name 'sum-product' comes from the Equation (2.2), which is performing summations and multiplications on messages.

Step 3) Compute messages at variable nodes and pass to check nodes,

$$L(q_{mj}) = L(q_j) + \Sigma_{m' \in C(m)\setminus\{m\}} R_{m'j}.$$

Step 4) The $a$ posteriori reliability messages are updated according to:

$$L(Q_j) = L(q_j) + \Sigma_{m \in C(j)} R_{mj}.$$
Step 5) Threshold the values calculated in each variable node to find a codeword. For every row index $j$:

$$
\hat{c}_j = \begin{cases} 
1 & \text{if } L(Q_j) < 0 \\
0 & \text{else}
\end{cases}
$$

(2.5)

If the codeword satisfies all the parity check equations or if the maximum number of iterations is reached, then stop. Otherwise, continue iterations from Step 2.

**Min-Sum algorithm:** Min-Sum algorithm is an approximation of the sum-product algorithm in which a set of calculations on the nonlinear function, $\phi(x)$, is approximated by a minimum function. Consider the update equation for $R_{mj}$ in the Sum-Product algorithm (Eq (2.2)). Fig. 2.2 illustrates the function $\phi(x)$, which is a mono-
tonically decreasing function for the values of $x > 0$. It is intuitive that the term corresponding to the smallest $\|L(q_{m'j'})\|$ value in Eq. (2.2) dominates the summation. Therefore:

$$
\Phi(\sum_{j \in N(m) \setminus \{j\}} \Phi(\|L(q_{m'j'})\|)) \approx \Phi(\Phi(\min_{j'} \|L(q_{m'j'})\|)) = \min_{j'} \|L(q_{m'j'})\|. \tag{2.6}
$$

The second equality follows from $\Phi(\Phi(x)) = x$. Comparing the Min-Sum algorithm to the Sum-Product algorithm, Eq.(2.2) in the Sum-Product algorithm is replaced by the following approximation in Min-Sum:

$$
R_{mj} \approx \prod_{j \in N(m) \setminus \{j\}} \text{sign}(L(q_{m'j'}) \times \min_{j' \in N(m) \setminus \{j\}} \|L(q_{m'j'})\|. \tag{2.7}
$$

Because of the above approximation, there is some degradation in the performance of the Min-Sum algorithm compared to the Sum-Product algorithm. The error correcting performance of the Min-Sum algorithm can be improved by either adding an offset to [56], or scaling the soft information [57]; the new variation of the algorithm can be called Modified Min-Sum algorithm.

**Modified Min-Sum algorithm** In the literature, it has been experimentally shown that scaling the soft information during the decoding step using the Min-Sum algorithm results in better performance. Scaling slows down the convergence of iterative decoding and reduces the overestimation error compared with the Sum-Product algorithm. Heo [57] showed that density evolution techniques can be used to determine the optimal scaling factor. He also showed that for a (3, 6) LDPC code a scaling factor
of 0.8 is optimal. Therefore, it is enough to replace the Eqn (2.3) in the Sum-Product algorithm with the following equation:

$$L(q_{mj}) = (L(q_j) + \sum_{m' \in C(j) \setminus \{m\}} R_{m'j}) \times \gamma,$$

(2.8)

in which $\gamma$ represents the scaling factor.

Layered Belief Propagation Algorithm

Layered belief propagation (LBP) algorithm was defined in [31]. This algorithm converges twice as fast as the original Sum-Product algorithm due to the optimized scheduling of reliability messages [58]. In this algorithm the PCM is viewed as a group of concatenated horizontal layers, where every layer represents a component code. The belief propagation algorithm is repeated for each horizontal ‘Layer’ and updated a posteriori probabilities are exchanged between layers. In this algorithm, the Eqs. (2.1, 2.2, 2.3, 2.4) are performed for each layer separately. Therefore, they are modified as following equations:

$$L(q_{mj}) = L(q_j) - R_{mj},$$

(2.9)

$$R_{mj} = \prod_{j \in N(m) \setminus \{j\}} \text{sign} \left( L(q_{m'j}) \right) \phi \left[ \sum_{j \in N(m) \setminus \{j\}} \phi \left( L(q_{mj}) \right) \right],$$

(2.10)

$$L(q_j) = L(q_{mj}) + R_{mj}.$$  

(2.11)

The layered belief propagation may also be modified to include the correcting offset [56]. As a result, the updating of check node messages in the $m^{th}$ row of the $k^{th}$
decoding iteration can be done as follows:

\[ R_{mj} \approx \prod_{j \in N(m) \setminus \{j\}} \text{sign}(L(q_{mj})) \times \max \left( \min_{j \in N(m) \setminus \{j\}} |L(q_{mj})| - \beta, 0 \right), \]

where correcting offset \( \beta \) is a positive constant. Hard decisions can be made after every horizontal layer based on the sign of \( L(q_j) \), \( j = 1, \ldots, n \). If all parity-check equations are satisfied or the pre-determined maximum number of iterations is reached, then the decoding algorithm stops. Otherwise, the algorithm repeats from Eq. (2.9) for the next horizontal layer.

### 2.1.5 Degree Distribution of LDPC Codes

In [59], an asymptotic analysis of LDPC codes under message-passing decoding (Density Evolution) was presented. [60] showed that by using density evolution technique, one can track the distributions of the messages exchanged in the iterative decoders for infinitely long codes. They presented the criteria for designing good irregular LDPC codes.

A polynomial \( \gamma(x) \) of the form

\[ \gamma(x) = \sum_{i \geq 2} \gamma_i x^{i-1} \quad (2.12) \]

is a degree distribution if \( \gamma(x) \) has nonnegative coefficients and \( \gamma(1) = 1 \). A code ensemble is defined with a degree distribution pair \((\lambda, \rho)\) in which

\[ \lambda(x) = \sum_{i=2}^{d_r} \lambda_i x^{i-1} \quad (2.13) \]

\[ \rho(x) = \sum_{i=2}^{d_c} \rho_i x^{i-1} \quad (2.14) \]
specify the variable and check node degree distributions. Here, \( \lambda_i(\rho_i) \) represents the fraction of edges originating from variable (check) nodes of degree \( i \). For example, for the \((3, 6)\) regular code we have \( \lambda(x) = x^2 \) and \( \rho(x) = x^5 \). The maximum variable degree and check degree is denoted by \( d_v \) and \( d_c \) respectively. If the code has \( n \) variable nodes, the number of variable nodes of degree \( i \) is then

\[
\frac{n \lambda_i}{\sum_{j \geq 2} \lambda_j/j} = \frac{n \lambda_i}{\int_0^1 \lambda(x)dx}.
\]  \(2.15\)

There are two major techniques in the literature for constructing and optimizing LDPC codes.

- Density evolution [60] or its approximation, Gaussian approximation [61].

- Extrinsic Information Transfer (EXIT) chart [62–66].

Both of these techniques are valid when designing an infinite-length LDPC code. Unfortunately there is still no known method for designing optimal finite-length codes. In the next section we give a brief overview of Gaussian approximation of the density evolution.

2.1.6 Gaussian Approximation of Threshold

In Gaussian approximation [61], the densities of the messages passed between variable nodes and check nodes are approximated with Gaussian densities. This results in significant complexity reduction in the density evolution algorithm. The main formula from [61] for calculating the thresholds of irregular LDPC codes is:
\[ m_u^{(l)} = \sum_{j=2}^{d_r} \rho_j \phi^{-1} \left( 1 - \left[ 1 - \sum_{i=2}^{d_l} \lambda_i \phi(m_{uo} + (i-1)m_u^{(l-1)}) \right]^{j-1} \right). \quad (2.16) \]

For \( 0 < s < \infty \) and \( 0 \leq t < \infty \), we define \( f_j(s, t) \) and \( f(s, t) \) as

\[
\begin{align*}
  f_j(s, t) &= \phi^{-1} \left( 1 - \left[ 1 - \sum_{i=2}^{d_l} \lambda_i \phi(m_{uo} + (i-1)m_u^{(l-1)}) \right]^{j-1} \right) \quad (2.17) \\
  f(s, t) &= \sum_{j=2}^{d_r} \rho_j f_j(s, t). \quad (2.18)
\end{align*}
\]

Using this, we can rewrite the above equation as

\[ t_i = f(s, t_{l-1}) \quad (2.19) \]

where \( s = m_{uo} \) and \( t_i = m_u^{(l)} \). The initial value \( t_0 \) is 0. The threshold \( s^* \) is the infimum of all \( s \) in \( \mathbb{R}^+ \) such that \( t_i(s) \) converges to \( \infty \) as \( l \to \infty \). For a detailed discussion of the Gaussian approximation the reader is referred to [61].

For punctured LDPC codes, Ha et al. [67] modified the Gaussian approximation to account for the punctured bits and puncturing ratios. In their analysis, eqn. (2.16) is replaced with the following equation:

\[
\begin{align*}
  m_u^{(k)} &= \sum_{s=2}^{d_r} \rho_s \phi^{-1}(1 - \frac{1}{(1 - \phi(k))^{s-1}} \left[ 1 - \sum_{j=2}^{d_l} \left\{ \lambda_j \phi \left( i m_u^{(k-1)} \right) \right\}^{j-1} \right]^{s-1}) \\
  &+ \lambda_j^{(1-s)} \sum_{i=0}^{j-1} (j-1) \chi_i^{(k)} \phi \left( i m_u^{(k-1)} + m_{uo} \right) \right\}^{s-1}) \quad (2.20)
\end{align*}
\]
in which

\[ e^{(k)} = \sum_{j=2}^{d_i} \lambda_j \pi_j^{(0)} (e^{(k-1)})^j \]

\[ \lambda^\pi(x) = \sum_{j=2}^{d_i} \lambda_j^\pi x^{j-1}. \]

\[ \lambda_j^\pi = \lambda_j \pi_j^{(0)}. \] (2.21)

For a detailed discussion on the Gaussian approximation of punctured LDPC codes, the reader is referred to [67].
2.2 Cooperative Communications

Cooperative communication takes advantage of the transmit diversity in wireless channels by emulating a multiple input multiple output (MIMO) system using single antenna transmitters. These distributed transmitters interact with each other to jointly transmit their information to a destination base station.

Wireless channels impose different amounts of attenuation on the transmitted signals depending on the physical phenomenon that the signal is passing through. By transmitting multiple independent versions of the signal transmit diversity can be achieved and effects of channel and fading can be removed. In a mobile environment, there are limitations in terms of power and size of the transmitter units that prevent the mobile units from supporting several transmit antennas. As a result, the mobile units can benefit from cooperation by sharing their antennas with other units to achieve diversity gain and increase the transmission rate or the battery life.

Several issues arise in a cooperative environment, such as relay selection and overall power and interference of the system. Other important issues are processing power, latency at the relay and the hardware overhead and architectural complexity of the relay node to support cooperation. In order for cooperation to be feasible, the extra hardware cost at the relay should be minimized. In addition, if the processing delay at the relay is large, the signal may get to the destination too late to be useful. Therefore, it may undermine the benefits of relaying. Considering that the nodes are mobile and run on batteries, minimizing the power spent at the relay on processing
and retransmission of the signal of other nodes is also desired.

Cooperative communication has several applications such as wireless cellular communication, ad-hoc networks, sensor networks, etc.

2.2.1 User Cooperation

Figure 2.3 shows a simplified version of user cooperation. In this figure, source (S) intends to transmit some data to the destination (D). If quality of the channel between S and D is good, then S transmits the data directly to the destination. If the channel between S and D is weak, then, S sends the information to the relay (R) in addition to D, and a relay re-transmits the signal to the destination possibly after some processing.

One of the main assumptions about cooperative communication is that the destination can distinguish between the signal that is received from the source or from the relay. One way is to separate these signals in time which means that in each time
slot either the source or the relay is transmitting. Another method is to transmit in different frequency domains or transmit orthogonal signals from source and relay.

Another issue is using full duplex radios that can transmit and receive signals at the same time. This is difficult in practice because transmitted signals can be up to $100\text{dB}$ stronger than the received signals, which could be beyond the isolation achievable by existing directional couplers. One solution can be time-sharing and using half-duplex relays that are more practical and less expensive. In this thesis, half-duplex relays are considered.

2.2.2 Related Work

Three terminal communication channels, namely relay channels, were introduced by Van der Meulen [68, 69]. He discovered lower and upper bounds on capacity of the relay channels. These bounds were improved in [70]. The interest in relay channels subsided until a few years ago, when we see increasing attention to both cooperation and the relay channels. The recent surge could be as a result of the major improvements in our understanding on the wireless environment, such as fading channels, discovery of turbo and re-discovery of LDPC codes, etc.

Other major contributions in this area are [71, 72], in which the authors proposed user-cooperation as a form of diversity in a mobile environment. Although for a different channel and user scenarios, the basic ideas behind relay channel and user cooperation are very similar. Cooperative diversity [3, 73] results when users cooperate
in order to utilize the spatial diversity to improve the reliability of communication in terms of the outage probability, bit error probability or increase in communication rate.

The outage probability and the asymptotic error probability performance were calculated in [74] for a two-user incremental redundancy coding scheme based on Gaussian code books. A practical two-user collaborating transmission based on rate compatible punctured convolutional codes is proposed in [75] and [76].

[2] utilizes limited feedback links to the transmitter nodes to maximize the throughput and minimize the outage [77, 78] in cooperation relay networks.

In recent years, researchers have combined the benefits of strong channel coding with cooperation. For example, [79] introduces cooperation diversity, in which users achieve maximum diversity by relaying each other’s data. The authors in [79] employed turbo codes with incremental redundancy in their analysis and showed its benefits. [80] finds capacity approaching turbo codes for relay channels in which relays work in full duplex mode.

A turbo-based coding scheme for relay channels is proposed in [81], which uses the topology of a distributed turbo code by placing RSC (recursive systematic convolutional) encoders at both the source and the relay and an interleaver at the relay to re-encode and interleave the signal received from the source. This method achieves an interleaver gain. Authors in [82–86] proposed LDPC code design for half-duplex relay channel. They optimized LDPC codes suitable for relay channels to perform
close to the theoretical limit.

[87] studies an incremental redundancy cooperative coding scheme. They propose a cluster-based collaborating strategy with one bit feedback. Other noteworthy references in the literature are [88–93].

The authors in [94] propose a different relay channel called the ‘relay without delay’ channel, in which the relay depends on the present transmission in addition to the past transmissions.

Types of Cooperation

Several types of user cooperation strategies exist depending on the processing performed by the relay on the information received from the source. The simplest strategy is Amplify and Forward [95], in which the relay receives a copy of a signal from a source node, amplifies the signal subject to its power constraint, and re-transmits the signal to the destination. The disadvantage of this protocol is that the noise is amplified in addition to the signal.

The next scheme is Decode and Forward [71, 72] in which the relay detects or decodes the received signal and re-encodes and forwards the new encoded signal. Decode and forward has been shown to have gain when the source is closer to the relay than the destination. There is a trade-off between the amount of computation that the relay performs on the signal and the delay that the destination sees because of the decoding and re-encoding of the signal by the relay, and also the power consumption
of the system.

Another scheme is *Estimate/ Quantize/ Compress and Forward* in which the relay compresses its received signal before retransmission [96–98]. Compression usually implies quantization, ideally taking into account the side information that is already available at the destination in the form of its received signal. Such quantization using side information available only at the destination is also known as Wyner-Ziv coding [99, 100]. The EF protocol approaches the capacity of the Gaussian relay channel as the relay-destination link grows stronger.

*Dynamic Relaying or hybrid* [95] refers to the scheme in which the source continues to transmit the signals directly to the destination when the fading coefficient between the source and the relay is small. And when the fading coefficient is larger than a certain threshold, it transmits the information to the relay first. The relay performs either amplify and forward or decode and forward on the signal.

*Incremental Relaying* exploits limited feedback from the destination to confirm the success or failure of the direct transmission. This scheme eliminates the unwanted repetitions of the signal specially for high rates. It can be viewed as an extension of the hybrid automatic repeat request (ARQ) to the relay context.

*Coded Cooperation* [75] integrates cooperation into channel coding. It works by sending different portions of the source's codeword through different paths, each of which carries redundant bits that can help the destination in decoding. It is important to note that there is no feedback in the system and everything is managed in the code
design.

In this document, we focus on the decode and forward scheme using low density parity check codes. Because of the error correction that happens at the relay, decode and forward can achieve very high gains. The drawback of this scheme is the processing time and power that is spent on the signal at the relay could potentially be very high. Optimizations or modification in the algorithms are needed to make them feasible in practice, which is the focus of this thesis.

2.3 Puncturing

In coding theory, the term ‘puncturing’ refers to the process of removing some of the bits from the codeword after encoding with an error-correcting code. Therefore, the code rate of the codeword is increased without a change in the trellis structure of the convolutional or turbo codes, or the parity check matrix of a LDPC code. Puncturing has a similar effect as encoding with an error-correcting code with a higher rate. However, the same decoder can be used for the original and the punctured code regardless of how many bits have been punctured. Therefore, the flexibility of the system considerably increases without significantly increasing its complexity.

2.3.1 Related Work

Rate compatible puncture convolutional codes [101] and rate compatible turbo codes [102–104] are generated from a mother code through puncturing.
[105] introduced the notion of rate-compatible LDPC codes and showed that higher rate codes can be derived by puncturing the mother code and lower rate codes can be produced by extending the parity check matrix. Authors in [106, 107] also generated lower rate codes by a progressive edge growth [108] technique. For LDPC codes, the puncturing techniques primarily focus on parity puncturing [109–111] and some on puncturing the information bits [112] or 'information shortening'.

In [67, 113–116], Ha et al. calculated asymptotic thresholds for an ensemble of punctured LDPC codes by modifying the density evolution with Gaussian approximation [61]. They optimized the puncturing patterns with respect to their threshold for infinite-length LDPC codes. They also proposed a procedure for determining the puncturing distributions for short length LDPC codes and showed that careful puncturing can provide good performance. They introduced the notion of $k$-Step Recoverable (or $k - SR$) for the punctured nodes, which is equal to the number of iterations that are required ($k$) for a punctured node to recover with a reliable message.

Authors in [117, 118] compare different puncturing patterns for a few different constructions of irregular LDPC codes from the same profile with similar variable node and check node degrees. They showed that if a puncturing pattern is optimized for a specific code, it can give catastrophic results if used on another code constructed from a similar profile.

A puncturing algorithm for block-type LDPC codes is proposed in [119]. They showed puncturing is equivalent to merging the check nodes connected to the punc-
tured bit and proposed a decoding algorithm for these punctured codes. Authors in [120,121] offered a few block-wise puncturing patterns based on maximizing the number of surviving check nodes of each punctured bit. Their algorithm works when the number of punctured bits is an integer multiple of the block size. In [122,123] generalized puncturing patterns for block-type LDPC codes are offered. However, in their methods, they are just puncturing the parity bits and do not puncture any information bits.

Semi-random puncturing patterns based on maximizing the average girth of the underlying Tanner graph of the punctured code are designed in [124]. By removing the short cycles, they compare their results with randomly generated punctured code and show the benefits. However, they do not offer any comparison with other puncturing patterns in the literature. Several researchers [67,109,125] have shown theoretically that punctured codes can be good candidates for achieving capacity.

A class of efficiently encodable rate-compatible (E2RC) LDPC codes is proposed in [126-128]. In E2RC codes, the parity check matrices have a favorable structure for puncturing and linear-time encoding. They showed that the E2RC LDPC codes have superior performance to other punctured LDPC codes. [129] continued their work to showed how to eliminate the high error floors associated with these codes. [130] offers a HARQ scheme based on shortened LDPC codes that reduces the computational complexity of the decoding at the receiver. Rate compatible puncturing of LDPC codes are studied in many papers such as [131,132] and [133].
A few papers talk about using puncturing in addition to cooperation. All of them are focused on using punctured convolutional or turbo codes. [88,134] talk about using punctured turbo codes in coded cooperation. In these papers, user1 transmits a punctured version of its codeword to user2. Upon successful decoding, user2 retransmits the bits that were punctured from the messages of user1 to the destination. The destination performs a full decoding on the two versions of the codeword received from both users by emulating distributed turbo coding. [87,135] extend these ideas to more users and talk about incremental redundancy cooperative coding using punctured turbo codes. [136] shows the benefits of coded cooperation with distributed punctured turbo codes and H-ARQ.

2.4 Simulation Environment

Throughout the thesis Monte Carlo simulations are used to validate the contributions. Unless otherwise stated, in majority of the simulations additive white gaussian noise (AWGN) channel with binary phase shift keying (BPSK) signalling is used. For the calculation of the bit error rate (BER) and frame error rate (FER), the simulations are repeated for enough experiments to get at least 100 frames in error. We have also assumed half-duplex relays that use time multiplexing with the source node to transmit their information.
2.5 Conclusions

In this chapter we presented the background and the literature review about LDPC codes and their decoding process and their architectures in the literature, Cooperative communications with decode and forward and also incremental redundancy and puncturing. In the next three chapters, we will discuss our results and contributions of the thesis in the areas of architecture and algorithm. We will present a flexible LDPC decoder architecture in the next chapter and show how to use this architecture in cooperative communications and how to make algorithmic simplifications to reduce the complexity of the cooperative systems in distributed decoding and puncturing in the chapters to follow.
Chapter 3

High Throughput Flexible Architecture for Low Density Parity Check Codes

Low Density Parity Check (LDPC) codes are one of the best error correcting codes that enable future generations of wireless devices to achieve higher data rates with excellent quality of service. This chapter presents a novel flexible decoder architecture for LDPC codes. The decoder supports irregular LDPC codes with twelve combinations of code lengths -648, 1296, 1944-bits and code rates $1/2, 2/3, 3/4, 5/6$ – based on the IEEE 802.11n standard. All codes correspond to a block-structured parity check matrix, in which the sub-blocks are either a shifted identity matrix or a zero matrix. A prototype architecture the flexible LDPC decoder has been implemented and tested on a Xilinx FPGA.

Generally, throughput and complexity of LDPC decoding depend on six major parameters: number of bits in a codeword or ‘block length’, ratio of number of information bits to codeword length or ‘code rate’, complexity of computations at each processing node, complexity of interconnections, parallelism level, and number of times that local computations are repeated, or number of iterations. There is a trade-off between performance of an LDPC decoder, complexity and speed of decoding. We will address these trade-offs throughout this thesis in more detail.
One of the main advantages of LDPC codes is the inherent parallelism in the decoding process which may lead to a very high decoding throughput if used properly. In this chapter, we present a novel high throughput LDPC decoder [8, 9, 137] that can achieve throughputs up to 736 Mbps. By using the knowledge gained from our earlier work on designing LDPC architectures for regular codes [32], we describe the design of a high throughput, flexible LDPC decoder that supports multiple block lengths (648, 1296, 1944 bits) and multiple code rates (1/2, 2/3, 3/4, 5/6) based on the IEEE 802.11n standard [4] for irregular LDPC codes. These codes are block-structured with profiles (refer to [26]) that provide error-correcting performance close to excellent fully random codes, such as the codes from [138]. In [10, 38], we designed additional codes and generalized the encoder/decoder architectures to support block lengths and code rates with finer steps.

The LDPC decoder presented in this chapter has a semi-parallel design and supports a family of Quasi-cyclic LDPC codes with small control and arithmetic logic overhead [8]. It can also switch between different code sizes and code rates on the fly without any need for recompilation or re-synthesis of the design. Furthermore, it uses layered belief propagation which converges twice as fast as the standard belief propagation algorithm, resulting in two times higher data throughput [139]. This decoder architecture is easily expandable to support other code sizes/rates.

Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs) are very suitable for designing LDPC encoders/decoders because
of the flexible number of processing units that can work in parallel rather than the limited number of processing units in Digital Signal Processors (DSPs) or general purpose processors (GPPs).

Every application has its own requirements and specific parameters. In order to decrease the design time and reuse the same hardware for many applications, a general, configurable, flexible architecture that supports several cases is of interest. This LDPC decoder should support a family of codes and should be able to switch between different code types seamlessly. For example, in wireless communication systems, or in user cooperation schemes, when the transmission channel is good, codes with fewer number of redundant bits or higher rates may be used, or, smaller block lengths may be used when there is less data to be transferred. Therefore, a multi-size, multi-rate LDPC decoder is a suitable choice for future generations of wireless devices.

In the following sections, we show that by taking advantage of the iterative decoding process and structure of the code matrix, we can increase the parallelism level of the architecture and increase the throughput of the decoder.

3.1 LDPC Decoder Architecture for Block-Structured Irregular LDPC Codes.

In this section, we describe the details of an LDPC decoder that supports irregular block-structured parity check matrices (PCMs) proposed for the IEEE 802.11n
wireless standard. Based on this standard, sub-block sizes are multiples of 27 and the code profiles are optimized to minimize the number of short cycles, and therefore achieve excellent performance. As an example, a PCM of a code with block length of 1296 bits and code-rate of $2/3$ is illustrated in Fig. 3.1. The PCM is partitioned into square sub-matrices of size $54 \times 54$ bits with at most one nonzero entry per row/column. Each sub-matrix is either a shifted identity matrix or a zero matrix. The black squares represent a shifted identity matrix and the white squares represent a zero matrix. The IEEE 802.11n standard supports block-structured irregular codes. Therefore, the PCMs for these codes contain a few zero blocks in addition to the shifted identity matrices. Existence zero blocks in the PCM leads to more complex memory addressing in the decoder for irregular codes compared to regular codes.
3.1.1 Scalable Decoder Architecture for Irregular Codes

In this section, we present the description of our configurable, flexible LDPC decoder implementation. Based on the standard and simulation results presented in section 3.1.4, the following features are employed: The layered belief propagation decoding algorithm is utilized in our design because of its fast convergence compared to the Sum-Product algorithm. This results in a similar error-correcting capability using only half of the decoding iterations. The parity check matrix is block-structured and irregular, which preserves the excellent performance of irregular random PCMs. It should be noted that designing an architecture for irregular codes is more challenging because it has to be flexible to support different row/column degrees. The decoder accepts block lengths of $N = 648, 1296, 1944$ bits as input, which corresponds to sub-block sizes of $S = 27, 54, 81$, respectively. It also supports four code rates: $R = 1/2, 2/3, 3/4, 5/6$.

Figure 3.2 illustrates a block diagram of our proposed LDPC decoder. Decoding is based on Eqs. (2.9), (2.10), and (2.11) described in Chapter 2. The decoder consists of processing units, memory blocks (RAMs and ROMs) for storage of messages, permuters to route messages from memory to the processing units, and a control unit. Decoding starts by reading a value that sets the block length and the code rate of the input codeword block. The decoder, uses this value to read appropriate values for the PCM from a number of ROM memories that contain Shift, Offset, Position and $w_r$ values, in which $w_r$ is the degree of each block-row in the parity check matrix.
These values are used to generate the addresses to read/write from/in the check and variable memories. The decoder also inputs the reliability values from the received signals from the channel and stores them in the variable memories. These values pass through the permuters to be routed to the correct decoding functional units (DFU) for processing. After each iteration of the decoding process, the results enter an 'early detection unit' (EDU) to check if the valid codeword is found. Also, the results are written back to the variable and check memories. The next layer/iteration starts when the decoder reads a new set of values from the variable memories. It should be noted that the early detection unit analyzes the decoded block in parallel with the main processing path. The decoder stops and outputs the resulting codeword when
one of the following two conditions is satisfied: if EDU detects a valid codeword, or a maximum number of iterations is reached. The control unit controls the flow of data among all the units. We will discuss each block of the architecture in more detail in the following sections.

Decoding Functional Unit Banks

The decoding functional unit banks are the main processing part of the architecture. They input the current values of check and variable messages and calculate new values. Number of parallel processing elements in these banks determines the parallelism factor of the design. For example, for the code in Fig. 3.1 with a block length of 1296 bits, 54 processing units may work in parallel without any data dependency restrictions. To support different code lengths, the architecture supports a collection of sub-block sizes of \( S = 27, 54, 81 \) bits. Hence, we divide the decoding functional units into three banks, each of which contains 27 DFUs. For a block of size 1944 bits, all three banks are used (81 DFUs), for a block of size 1296 bits, two of the banks are used (54 DFUs), and for a block of size 648 bits only one bank is employed. The decoding functional unit (DFU) inputs \( w_r \) values from the check memories and \( w_v \) values from the variable memories, and generates new variable and check messages based on Eqs. (2.9), (2.10) and (2.11).

Fig. 3.3 shows a block diagram of a decoding functional unit. The values of variable and check messages enter this unit and the difference is calculated based on
Figure 3.3: The decoding functional unit block diagram.

Eqn. (2.9). Sign and magnitude of inputs are separated and pass through separate computation paths based on Eqn. (2.12). The serial Min-Sum unit block inputs the absolute value of the variable messages and finds the two minimums among the $w_r$ values and the relative index of these values compared to the first input. Then, the offset value is subtracted from the minimums. The intermediate output corresponding to each input is the minimum of all the other values.

The sign of each output is the multiplication of the signs of all the inputs other than the input corresponding to this output. Then, the previous check node values are subtracted from these and the variable node values are added to the results to generate the final outputs ($w_r$ output values) based on Eqn. (2.11). The outputs of all the S-DFUs are concatenated and stored in one address of the variable memories. It is important to note that we do not permute back these values; variables are stored in the shifted order. The permuter in the next layer uses values from the Offset
ROM instead of the original shift values. This way we eliminate the need for another permuter, which saves 8% of the total area.

**Serial Min-Sum unit**  This unit inputs $w_r$ values and finds the two smallest values. Depending on the requirements on the architecture, this unit can be designed in two different ways: 'Parallel-Input' (PI) or 'Serial-Input' (SI). The PI generates the final results very fast by using multiple processing units in parallel. This approach is suitable for a decoder that supports regular LDPC codes with a fixed number of inputs that needs a predetermined number of processing units. Since our architecture supports irregular codes, the number of nonzero sub-matrices in each row of the PCM is different for each layer. This means that the number of inputs for the Min-Sum unit varies based on the $w_r$. 

Figure 3.4: The serial Min-Sum unit block diagram.
Figure 3.5: The flexible permuter block diagram.

In the irregular PCMs supported by this architecture, number of inputs to the min-sum unit varies from 7 to 21 for different code rates. In order to support variable number of inputs, we have designed a 'serial' Min-Sum unit. Although this unit runs slower than the parallel version, it has the flexibility of accepting any number of input values in a serial fashion using only one input port. In this way, the hardware can support irregular codes of different rates and sizes. The controller marks the beginning and end of the input sequence. The Min-Sum unit finds the two smallest numbers and their location in the sequence compared to the first input. Fig. 3.4 shows the block diagram of the serial min-sum unit.

**Flexible Permuters**

One of the main challenges of the LDPC decoder is 'routing' the messages from the memories to the correct processing units as quickly as possible. For a fixed decoder
that supports a single block length/code rate, routing can be done by utilizing address
generators, split/merge units and routing wires. For a more general architecture,
routing can be done with a network of multiplexers to route the signals according to
the shift values. This multiplexer network is called a 'permuter'.

In this architecture, permuters are used to shift a block of $S$, $b$-bit numbers to
generate the correct addressing based on the values in the parity check matrix. For ex-
ample, a shift of $s$ means that the order of the outputs should be $(s+1), \ldots, S, 1, \ldots, s$
instead of $1, \ldots, S$. Analysis was done on the structure of permuters to determine
the best structure. We tested permuters of size 81 using different sizes/combinations
of multiplexers, and selected the one with the smallest area and highest speed, which
is designed with four levels of $4 : 1, 4 : 1, 4 : 1, 2 : 1$, multiplexers with $b$-bit in-
puts/outputs.

Since permuters occupy a significant portion of the decoder area, instead of having
three permuters to support different values of $S$ (corresponding to different code
rate/sizes), we have designed a flexible permuter of size 81, which can be used to
permute any of the sub-block sizes. This flexible permuter is made by adding a layer
of multiplexers to the original permuter of size 81 and selecting the proper signal for
each case as shown in Fig. 3.5.
Memory Organization

There are several ROM/ RAM blocks in the system. These blocks are divided into different banks for check and variable messages and a few ROMs that store the parameters to regenerate the various parity check matrices. To be able to read/write one full sub-block matrix per clock cycle, the check memory and the a posteriori memory need to be organized in an appropriate manner.

Three memory blocks are used to store the check messages. Organization of the check-node memory is illustrated in Fig. 3.6. In order to increase the throughput of the decoder and take advantage of the parallel processing, we use packed storage of the check and variable messages. \( S \) messages are concatenated and stored in a single memory address that can be read in a single clock cycle. Permuters are used to split and route each single message to the corresponding DFU unit. Also, we have divided these memory blocks into three modules that accept the same address (to avoid extremely large memory word lengths). Each of these modules packs and
stores 27, b-bit values from each sub-block matrix per address. When \( b = 8 \) bits, each word in the memory is \( 27 \times 8 = 216 \) bits. These values correspond to check outputs for the DFUs numbered from 1 to 27, 28 to 54, and 55 to 81, respectively. In this way, by reading from a single address from three memory blocks, all of the 81 check values are ready for concurrent processing.

For the largest codeword size of 1944 bits, all three check memory sub-modules are used, while only two or one module are used for codeword sizes of 1296 or 648, respectively. The unused check memory modules may be turned-off. The depth of the check memory sub-modules depends on the number of layers and number of non-zero sub-block matrices per layer (row connectivity degree). The largest depth for code rate of 1/2 is 96 since there are (in average, because of the code irregularity) eight non-zero sub-block matrices per layer and there are 12 layers. Addressing of check messages is simple since the memory locations are always accessed in an increasing order through the use of a hardware address increment unit.
Using the same packed storage concept, three dual port memories are used to store the variable messages. Organization of the memories that store the variable values is illustrated in Fig. 3.7. The variable values are also grouped and stored in a single memory address (groups of 27, 8-bit numbers). During the initialization step, these messages are stored in the original order of the inputs. They are permuted based on the elements of the PCM to route to the correct DFU for processing. Depending on the block sizes, the variable memories can be divided into three or more modules.

We should note that because of the special structured design of the LDPC codes there is no memory access conflict neither in the check memories nor in the variable memories in our decoder architecture.

**Packed storage of multiple PCMs** Our flexible architecture supports \( n = 3 \) block lengths and \( r = 4 \) code rates which results in \( \eta = n \times r = 12 \) combinations. All of the \( \eta \) parity check matrices that correspond to different code sizes/rates should be stored in read only memories (ROM). Since the PCMs are large, sparse matrices, they may require very large memories if stored directly. However, by taking advantage of the structure of the PCMs, it is possible to store a few parameters and regenerate the matrix when needed. Considering the structure of the PCM, we can rebuild it by knowing a few parameters: block length, sub-block size, number of nonzero sub-matrices in each layer of the PCM, position of the nonzero blocks and the shift value to generate the shifted identity matrices. The PCM can be regenerated on the fly using counters and permuters.
We have added another value to these parameters, namely 'offset' values. These values are the shift values with regard to the previous shift of the same layer (see Fig. 3.1). For example, if the shift value for layer \( k \) is \( s_k \) and the shift value for the next layer is \( s_{k+1} \), the offset \( o_{k+1} \) will be equal to: \( o_{k+1} = s_{k+1} - s_k \). The decoder uses the value of \( o_{k+1} \) in the permuter to route the messages.

**Early Detection Unit**

Early detection unit checks the output of each iteration to detect if the correct codeword is found. Usually, two sets of tests are performed after finishing the decoding of each layer. First, the early detection unit checks to see if all the parity check equations are satisfied \( (H_i \times c = 0) \). Then, the EDU compares the signs of the updated variable messages with the previous values of these messages and checks if the signs have changed: \( (\text{sign}(L(q_j^{(k,i)})) \times (\text{sign}(L(q_j^{(k,i-p)}))) > 0 \).

Outputs of a layer are valid if all the parity check equations are satisfied and there is no sign change in the results. It is required that \( L \) (total number of layers) consecutive layers satisfy these two constraints even if they belong to two consecutive iterations. Then, decoding stops and the resulting codeword is sent out of the decoder.

**Controller**

The controller block controls flow of the messages into and out of different blocks throughout the decoding procedure. The controller generates enable/reset and all the hand-shaking signals necessary for correct operation of the decoder. It also controls
the counters that generate addresses for ROM and RAM memories. The controller inputs the values of block size and code rate and also reads the number of nonzero blocks in each layer \((w_r)\) from the \(w_r\) ROM. Based on these parameters, this unit controls the flow of data to the DFUs, Min-Sum units, permuters and other blocks in the system. The main challenge of the controller is to keep track of the groups of \(w_r\) messages throughout the process and read/process/write them at the correct time.

3.1.2 Hardware Overhead

To support the highest data rates for wireless networks, this decoder is designed to support the largest block length \((N = 1944)\). For smaller block lengths, some of the DFU units and memories are unused and can be turned off/disabled using clock gating. Because of the data dependency, it is not possible to use a number of DFUs greater than the sub-block size. This is a limitation that is imposed by the PCM structure and the decoding algorithm. Another overhead in the design comes from storing twelve PCMs in the memories corresponding to each combination of the code rate/size, which is almost 22 Kbits. There is also 19% overhead for the flexible permuter compared to a fixed permuter of size 81.

3.1.3 Hardware Implementation of LDPC Decoder

Two prototype architectures for the LDPC decoder have been implemented utilizing System Generator from Xilinx Inc. and targeted to a Xilinx Virtex-4 \(xc4vlx60\) FPGA. These cases include:
• LDPC decoder with 7-bit messages (Dec7)

• LDPC decoder with 8-bit messages (Dec8)

Table 3.1 presents the utilization statistics of these two architectures. A clock frequency of 160 MHz is achieved for both of these designs after place and route.

Table 3.1 : Design statistics for the flexible irregular LDPC decoder on Virtex4-xc4vfx60 FPGA.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Dec7</th>
<th>Dec8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>11,328</td>
<td>12,633</td>
</tr>
<tr>
<td>FFs</td>
<td>12,368</td>
<td>13,823</td>
</tr>
<tr>
<td>LUTs</td>
<td>17,104</td>
<td>19,265</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>87</td>
<td>87</td>
</tr>
</tbody>
</table>

The proposed decoder architecture is also synthesized for a Chartered Semiconductor 0.13μm, 1.2V, CMOS technology using the BEE/Insecta design flow [140] and Synopsys tools. The Chartered memory compiler was used to generate efficient RAM and ROM blocks. Table 3.2 shows the area occupied by each part of the decoder with 8-bit messages in square millimeters. This architecture runs at a maximum clock speed of 412 MHz and consumes 502 mW of dynamic power (estimated using Design Compiler). The total area for the decoder is 2.2036mm² for the 7-bit decoder (Dec7) and 2.4928mm² for the 8-bit decoder (Dec8).
Table 3.2: ASIC design statistics for the Dec8 flexible LDPC decoder.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Area $mm^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Banks</td>
<td>1.1925</td>
</tr>
<tr>
<td>DFU Banks</td>
<td>0.939</td>
</tr>
<tr>
<td>Permuters</td>
<td>0.289</td>
</tr>
<tr>
<td>Control</td>
<td>0.0029</td>
</tr>
<tr>
<td>LDPC Decoder</td>
<td>2.4928</td>
</tr>
</tbody>
</table>

3.1.4 Decoding Throughput and Performance

Figure 3.8 shows the frame error rate (FER) performance of the implemented decoder for a 2/3-rate code with a code length of 1296 bits and the layered belief propagation decoding algorithm. Performance results for 6, 7 and 8-bit fixed point arithmetic precisions and floating point are shown. Both 7 and 8-bit fixed-point precisions have small loss compared to the floating point version.

We have estimated the decoding throughput (considering the information bits) based on the average number of decoding iterations required to achieve a FER of $10^{-4}$, while the maximum number of iterations is set to 15. The clock frequency is 160 MHz for the decoder running on the FPGA and 412 MHz for the ASIC implementation. Figure 3.9 shows the average throughput as a function of code rate and codeword size for the decoders. As an example, for a block length of 1944 and a rate 5/6 code, the
Figure 3.8: FER for irregular block-structured PCM (R=2/3, N=1296, MaxIter=15, layered belief propagation).

decoder implemented on the FPGA achieves an average throughput of 292 Mbits/sec and the ASIC version achieves 736 Mbits/sec. Average FPGA latency of the decoder is between 5 and 11\,\mu\text{sec} for different block lengths/rates while the average ASIC latency is between 2.2 and 4.5\,\mu\text{sec}.

The flexible decoder for irregular codes of rate 1/2 and block length of 1296 bits achieves a throughput of 56 Mbps, and the decoder for code rate 1/2 and block length of 1944 bits achieves a throughput of 84 Mbps.
Figure 3.9: Average decoding throughput for different code rates and codeword lengths for the flexible irregular architectures synthesized for FPGA and ASIC based on the average number of iterations.

3.1.5 Conclusions

We presented the design and hardware implementation of a flexible architecture for structured irregular LDPC codes. This decoder supports a family of code sizes and rates and can switch between different cases on the fly. The decoder uses serial computation units, which enhances the flexibility with the cost of increased latency. There is also some overhead in the memory and area usage of this decoder to support the 12 combinations of the code sizes and code rates. This decoder is suitable for applications that require scalability. The decoder has a general structure and can support both regular and irregular codes. It can also be extended to support other code families. The decoder is implemented on an FPGA and is synthesized for ASIC.

We present cooperative communications with distributed decoding in the next
chapter. We will show that our flexible LDPC decoder architecture can effectively be used in cooperation. Since the relays are chosen based on the current topology of the system, the relay should be able to support a family of the codes so that it can be paired with a number of source nodes. We will discuss this in more detail in the next chapter.
Chapter 4

Distributed Decoding in Relay Channels

In this chapter, we present a novel relaying strategy called distributed partial decoding. This strategy can be viewed as a variation of the decode and forward relaying with a difference that the relay partially decodes the signal, re-transmits it to the destination, and the destination continues the decoding. By distributing the decoding process between the relay and the destination, the relay uses less processing power and less time and requires smaller hardware to support cooperation compared to the original decode and forward strategy. The distributed partial decoding strategy is very suitable for practical applications, in which relays are battery-operated (such as handsets) and do not want to use all of their battery power on relaying data of other users.

It should be noted that throughout this work, we assume that the relay selection procedure is performed previously, and the best relay is chosen. There is a large body of research dealing with the relay selection [1, 141, 142]. We assume that one of the common strategies is used to choose the best relay.
Figure 4.1: Half-duplex relay channel. In the first half of the time slot, the source broadcasts its information. In the second half, both source and relay send some information to the destination.

4.1 Cooperative Communications

The terms 'cooperative communications' or 'relay channels' refer to a communication system in which neighboring nodes help each other in the transmission. Therefore, one of the users that does not have any data to send to the destination, acts as a relay and helps to transmit the data of other users. Cooperative communication has attracted major attention in the field of wireless communications in recent years. Most of the wireless devices operate in stand-alone fashion communicate directly with a base station. These devices are in idle mode for long periods of time and are only used for short periods resulting in under-utilized radios and inefficient use of network resources.
4.1.1 System Model

In this thesis, we assume a three-node communication link between a source, a relay and a destination node. Although our results can be generalized to two or more relays in the network, to simplify the discussion, we focus on one relay node. The relay is assumed to be half-duplex, which is either receiving or transmitting the data at each time instance. This restriction comes from practice because it is extremely difficult and expensive to build a relay that can work in full-duplex mode. The problem comes from the fact that the signal that is being transmitted is generally about $100 dB$ stronger than the received signal and the relay has to have proper shielding to eliminate the effects of the transmitting signal on the received signal.

We assume that the cooperative nodes work in two consecutive time slots $t$ and $t' = 1 - t$. In the first time slot $t$, a source node operates in the broadcast (BC) mode and transmits the data to both relay and destination nodes. In the second time slot $t'$, the source and the relay work in multiple access (MAC) mode. During the MAC mode, the destination receives two signals, one from the source and one from the relay. Fig. 4.1 illustrates the graphical representation of the relay channel in BC and MAC modes of operation.

Throughout this document, the channels between the cooperating nodes are assumed to be additive white gaussian noise (AWGN) channel and the modulation is assumed to be binary phase shift keying (BPSK) unless otherwise stated. To be fair in our comparisons with the direct link, we assume that the total transmission
power of the system is fixed. The total transmission power for the source in direct transmission is equal to the total power used by both the source and the relay during cooperation. Therefore, we assume a global power constraint as follows:

$$tP_{SC} + t'(P_{SMAC} + P_{RMAC}) \leq P,$$

(4.1)

in which $P_{SC}$ is the source transmission power in the BC mode, $P_{SMAC}$ and $P_{RMAC}$ are the source and relay transmission powers in the MAC mode, and $P$ is the total system transmission power. This assumption does not limit our results and in cases where the relay can add extra power to the system, using the relay may be even more beneficial.

We assume the relay lies on a direct line connecting the source and the destination, with normalized distance of $0 < d < 1$ between the source and the relay and the distance $0 < 1 - d < 1$ between the relay and the destination (See Figure 4.2).

Consider a data block of $K$ bits $(x_1, x_2, ..., x_K)$ that is encoded with an error correcting code, such as LDPC, turbo code, etc to generate the codeword $(c_1, c_2, ..., c_N)$ of length $N$ bits. The coded sequence is modulated and transmitted to both the relay
and the destination. The received signals at each of the nodes in the first and second
time slots (BC and MAC modes) are as following:

\[
Y_{SR_1} = h_{SR} m_1 + n_{SR_1}, \quad (4.2)
\]
\[
Y_{SD_1} = h_{SD} m_1 + n_{SD_1}, \quad (4.3)
\]
\[
Y_{SD_2} = h_{SD} m_2 + n_{SD_2}, \quad (4.4)
\]
\[
Y_{RD_2} = h_{RD} m'_1 + n_{RD_2}, \quad (4.5)
\]
\[
Y_{D_2} = Y_{SD_2} + Y_{RD_2}
= h_{SD} m_2 + h_{RD} m'_1 + n_{D_2}, \quad (4.6)
\]

in which subscripts 1 and 2 refer to BC and MAC modes of operation, \( m_i \) is the
modulated signal, \( m'_1 \) is the message at the relay after decoding, re-encoding and
remodulating, \( n_{ij} \) is the Gaussian noise with zero mean and variance \( \sigma^2 \), \( h_{ij} \) is the
channel realization between nodes \( i \) and \( j \), \( i,j \in \{S, R, D\} \), \( \gamma_{ij} = |h_{ij}|^2 \).

The channel gain for the SD, SR and RD links are given by:

\[
\gamma_{SD} = 1, \gamma_{SR} = \frac{1}{d^\alpha}, \gamma_{RD} = \frac{1}{(1-d)^\alpha}
\]
in which \( \alpha \) is the path-loss exponent [143] and typically ranges from 2 (e.g., free-space
propagation) to 4 (e.g., typical rural areas). We assume \( \alpha = 2 \) in this thesis; larger
values of \( \alpha \) increase the attenuation of the signal by the channel, which makes relays
and our results more favorable.
4.1.2 Experimental Setup

We assume a number of scenarios in this section consisting of direct link transmission or cooperative communications with different schemes and present the simulation results for each of them. The received power at each receiver is:

\[ P_{SR} = P_{SBC} \cdot \gamma_{SR} \]

\[ P_{RD} = P_{RMAC} \cdot \gamma_{RD} \]

\[ P_{SD1} = P_{SBC} \cdot \gamma_{SD} \]

\[ P_{SD2} = P_{SMAC} \cdot \gamma_{SD} \]

in which \( SD1 \) and \( SD2 \) refer to the signal that the source transmits in the first or second intervals to the destination. We assume that the total power of the system is fixed to be fair in the comparisons with the direct path scenario.

Figure 4.3 illustrates a general block diagram of a cooperative communication system using decode and forward strategy. As described in section 4.1, in the decode and forward strategy, the source encodes the signal with any channel coding scheme such as low density parity check (LDPC) codes, turbo codes, etc. The relay decodes the signal and re-encodes it with either the same code or a new code. Then, it transmits the new codeword to the destination. The destination utilizes both copies of the signal received from the source and the relay by combining and using them in the LDPC decoder. We assume AWGN channels between SR, RD, and SD nodes with pathloss.
In the simulation for the direct link, we omit the relay link in figure 4.3 and only consider the source and destination nodes. The direct link is the baseline for all the comparisons. In this case we assume that $t = 1$ and $\tilde{t} = 0$ in Eq. 4.1. We report the simulation results for the maximum number of decoding iterations equal to 10 and 20. We present simulation results for decode and forward relaying and some variations of this scheme and describe ways to reduce complexity of this scheme. We compare the performance of decode, no encode and forward(DNEF) with Decode, encode and forward (or the original decode and forward) and show that in some cases eliminating the encoding step at the relay is beneficial in reducing system complexity without performance degradation.
Decode, No Encode and Forward: DNEF

In decode, no encode and forward, a source node encodes the data with $H_S$ in broadcast mode and sends it to both $R$ and $D$. When a relay receives the codeword, the relay decodes the codeword for $i$ decoding iterations and transmits the resulting codeword to the destination in multiple access (MAC) mode. Note that the relay does not re-encode the signal after the decoding. However, the relay keeps final values of the parity bits after decoding and transmits the codeword from the final step of decoding before the parity bits are removed from the codeword. So, the relay transmits the decoded codeword without re-encoding to the destination. This idea was proposed in [144] for Hamming codes, showing the benefits in terms of delay and power consumption of the relay. We show that LDPC codes of larger sizes can benefit from this scheme too. One of the advantages of decode, no encode and forward is that the relay does not require specialized hardware for encoding, and spends less time and power processing data of other users.

We assume that the source is silent in MAC mode and the destination only receives one copy of the signal from the relay in the second time frame. We also assume that $t = \bar{t} = 1/2$, meaning that the two time intervals for BC and MAC modes are equal. The destination receives two different copies of the signal from the source and the relay, and performs Maximal Ratio Combining (MRC) on the received signals. The destination processes the two received copies of the signal from the source and the relay.
Figure 4.4: Decode, encode and forward (DEF) strategy vs. decode, no encode and forward (DNEF) strategy.

When the destination combines the two received signals and decodes the combined signal, decode and forward strategy is used. If the destination ignores the received signal from the source and decodes the signal received from the relay, it will be similar to two-hop communication with the difference that relay decodes and re-transmits the signal.

Decode, Re-encode and Forward: DEF

Decode, re-encode and forward is very similar to decode, no encode and forward, with a difference that the relay re-encodes the codeword after decoding and before transmitting it to the destination. The relay re-encodes the signal with a parity check matrix either similar to the one used by the source or a new parity check matrix. Using
a similar PCM as the source for encoding has benefits when there are still errors in
the parity bits after the relay decodes the codeword received from source. The old
parity bits are discarded and a new codeword is calculated at the relay. Using a new
PCM for encoding may be useful when the transmission time/channel quality for RD
link is different from the transmission time/channel quality for SR.

Figure 4.4 illustrates the difference between the DNEF and DEF relaying strate-
gies. This figure shows an LDPC codeword that passes through different blocks of a
cooperation system such as in figure 4.3.

4.1.3 Timing Diagrams

Figure 4.5 shows a timing diagram of the decode and forward (i.e., DEF) relaying
strategy. The relay is half duplex, therefore, each time-slot is divided into two half-
slots. The first half-slot is used for receiving the signal by the relay and the second
half-slot is used for transmitting the signal.

At the beginning, a source node transmits a packet to the relay and the destination
(Transmit-BC1). Both the relay and the destination start to detect the symbols and
buffer them in their memories as soon as they receive beginning of a packet. In the
second half-slot, the source sends some extra parity bits to the destination (the relay
does not need this extra information). At the same time, the relay starts to decode
the first packet (Decode-BC1) using an LDPC decoder. After a few iterations, the
result of decoding is ready. Next, the encoder starts encoding the information bits
Figure 4.5: Timing diagram of Decode and Forward Relaying. Notice that the relay has limited time to decode/re-encode and transmit the received signal.

(potentially with a different code) and transmits the new codeword to the destination.

During the second half slot, the destination operates in a Multiple Access (MAC) mode. The destination receives a copy of the signal from the source (Receive-MAC1) and a copy from the relay (Receive-BC1r), which is a transformed version of the first codeword originally sent from the source during the first half slot. Next, the destination combines the signals and decodes the packets received from the source and the relay and generates the final result.

While designing a practical cooperative system, the first obstacle that presents itself is due to the assumption in information theoretic analysis that processing at the relay takes no time. Based on an FPGA implementation of an LDPC encoder/decoder, a packet with a block length of 1296 coded bits in addition to a preamble and training takes $140\mu s$ to be transmitted. There is a $10\mu s$ latency for
Figure 4.6: An alternative timing diagram for Decode and Forward Relaying - More processing time at the relay.

detection. Decoding with 10 iterations takes about 80μs and a simple re-encoding needs an extra 10μs. The above numbers indicate that the processing time at the relay is not negligible at all, and can nullify the gain due to cooperation if the source broadcast and the relay transmission are spaced apart by a period of only processing and no communication. One way to overcome this problem is decreasing the processing time at the relay as much as possible. The idea of distributed decoding serves this purpose and reduces the decoding time at the relay considerably. Utilizing the decode, no encode and forward, also saves some more time by eliminating the need to re-encode the LDPC code.

A timing diagram for the DNEF relaying strategy is similar to Fig. 4.5. The only difference is that the blocks related to encoding at the relay (Encode BC1r, Encode
BC2r) are eliminated which enables the decoding and retransmission to spend more time for their tasks.

Another way to overcome this problem is by employing a schedule as illustrated in Figure 4.6 that starts with two consecutive broadcasts followed by a multiple-access, instead of a single broadcast followed by a multiple-access. After the initialization, broadcast and multiple-access modes alternate in the usual manner. One advantage of starting with two broadcasts is that when the relay is receiving the second broadcast transmission, it gets some time to process the first broadcast, so that it is ready to transmit its first multiple-access packet as soon as the second broadcast ends. Note that the length of a transmission, 140\mu s in this case, satisfies the requirement of being larger than the total processing time. This schedule takes advantage of the fact that although the relay cannot simultaneously transmit and receive (half-duplex), it can surely process information in parallel with transmission or reception. The gain in throughput due to this schedule comes with the overhead of double the amount of buffers and more difficult scheduling/timing of the system. The buffers are necessary to store the currently received signals while previously buffered signals are being processed.

It should be noted that Fig. 4.5 and the above timing estimates are based on the assumption that one OFDM frame contains just one LDPC block (for simplicity of description). If each OFDM frame contains two or more LDPC blocks, then the detector and the decoder can start processing as soon as they receive the first block.
Figure 4.7: Bit error rate vs. signal to noise ratio for different locations of a relay with respect to the source, Block length=1296 bits, code rate=1/2, $\alpha = 2$, DNEF.

This scheme has less overhead (in terms of preamble and training) and hence higher throughput and lower overall latency.

4.1.4 Simulation Results—Decode, No Encode and Forward, DNEF

Figure 4.7 shows the bit error rate vs. signal to noise ratio for the decode, no encode and forward scheme with maximal ratio combining at the receiver. The LDPC codes used in simulations are based on IEEE 802.11n standard. The LDPC codes have block length of 1296 bits and code rate of 1/2. Layered belief propagation decoding algorithm is utilized in both the relay and the destination. This figure illustrates the
decoding performance for different locations of relay \( 0 \leq d \leq 1 \) with respect to the source node. It is assumed that half of the total power \( (P) \) is used by the source and half of it is used by the relay. Simulation results show that the case of \( d = 0.4 \) has the best performance over all other positions of the relay. The results also show that position of the relay plays an important role in the system performance.

Bit error rate of the direct link with maximum of 20 decoding iterations is also shown for comparison. The figure shows 2.5 dB improvement for the BER of \( 10^{-4} \) over the direct link for \( d = 0.4 \). Also, although in this curve, both relay and destination use the same LDPC code, if we increase the number of decoding iterations at the destination, performance gain is minimal compared to the case that both relay and destination decode with 10 iterations. Therefore, distributing the decoding resources between the relay and the destination results in 2.5 dB gain compared to direct-link.

**Effect of Power Distribution**

A comparison between different power schemes is depicted in Figure 4.8. This figure shows the \( Eb/N0 \) values for a bit error rate of \( 10^{-4} \) for different locations of the relay and different power schemes. In this figure, the total power \( P \) used by the source and the relay is fixed, however the ratio of power used at the source or at the relay differs for different cases. It can be seen that if we change the power available at the source from \( P/4 \) to \( 3P/4 \), and change the power available at the relay from \( 3P/4 \) to \( P/4 \), the best location for the relay moves from \( d = 0.3 \) to \( d = 0.55 \). These results
Figure 4.8 : Effect of power distribution on the performance:(1296,1/2) LDPC code, \( \alpha = 2 \), DNEF

are intuitive because the more power we put on the source, the farther the data can be transmitted without error and hence the relay can be further away from source.

One important fact to note here is that by changing the power by 50%, the best location for the relay changes only 25%. This shows that cooperation is not very sensitive to power, which means that a coarse power management scheme having 4 to 8 power levels is enough for cooperation in practice.

One point that can be taken from these simulations is that for maximizing the gain through cooperation, a power management protocol needs to know the relative distance of the S, R and D nodes to assign the best power ratio to the source and the
Figure 4.9: Bit error rate vs. signal to noise ratio. Block length = 1296 bits, effect of code rate on the performance of the direct link.

relay node. This can also be seen as another parameter in relay selection. Depending on the amount of power available at the source and at the relay, one can choose the best relay possible.

Effect of Code Rate

In order to see the effect of the code rate on the performance of the LDPC decoder, some baseline simulations were performed on the direct link for the LDPC code of block length 1296 bits and several different code rates such as 1/2, 2/3, 3/4, 5/6. The results are shown in Figure 4.9. It can be seen that the more redundancy in a
Figure 4.10: Effect of code rate on the relay performance. (1296,1/2), $\alpha = 2$, DNEF.

code (i.e., the lower the code rate), the better the performance. A similar pattern is seen in Figure 4.10 for cooperative communications.

A comparison between different code rates is depicted in Figure 4.10. It shows that the code rate is not a determining factor of the shape of these curves and it just shifts them vertically (as expected), meaning that the higher code rate, the higher SNR is required for the same performance. This shows that when the links are stronger, we can use higher code rates to increase the throughput.
Figure 4.11: Bit error rate vs. signal to noise ratio. Similar codes for the SR, RD links, Relay: Decode, re-encode and forward

4.1.5 Simulation Results—Decode Re-encode and Forward, DEF

DEF with Homogeneous Codes at Source and Relay

To compare the results of the proposed scheme with the existing schemes in the literature, we have simulated the decode, re-encode and forward (i.e., Decode and Forward) strategy in which the relay decodes the codeword received from the source using LDPC decoding and re-encodes it using the same code. At the destination, two scenarios can be applied. In the first scenario, the destination employs two separate decoders for decoding the signals received from the source and the relay, and combines the soft outputs of the decoders to find the final decoded information bits. In the
Figure 4.12: Different code rates at the source and the relay with equal number of information bits. Source: (1944,1/2), Relay:(1296,3/4). Relay: decode, re-encode and forward, Equal power.

The second scheme, the destination combines the two signals using MRC before decoding, and jointly decodes the combined signals. The results are illustrated in Fig. 4.11. This figure shows that when two decoders are utilized at the destination, the best performance is when the relay is halfway between the source and the destination \((d = 0.5)\). When the destination jointly decodes the signals, the best location is at \(d = 0.4\). This scheme is also less sensitive to the distance when the relay is close to the source because it takes advantage of the diversity better than the first scenario.
DEF with Heterogeneous Codes at Source and Relay

Figure 4.12 shows the simulation results for heterogeneous codes at the source and the relay. In these simulations, the source uses an LDPC code with block length 1944 bits and code rate 1/2. The relay decodes the received codeword and re-encodes it using an LDPC code with block length 1296 bits and code rate 3/4. Note that both of these codes have equal numbers of information bits (972 bits). The relay encodes the block with a higher rate code with less redundancy. Hence, the number of parity bits and block length is smaller. The figure 4.12 shows that the heterogeneous case loses some performance compared to the homogeneous case, but the transmission time is decreased by \((1944 - 1296)/1944 = 33\%\). A more detailed comparison is presented in figure 4.13.

4.1.6 Comparison of DEF and DNEF

A comparison between the different decoding strategies is illustrated in Fig. 4.13. This figure shows four curves: The first curve (with triangles) corresponds to the decode and forward without encoding (DEF) for a (1296, 1/2) LDPC code. In this case the destination receives two separate signals from the source and the relay, combines them using MRC and jointly decodes them at the destination.

The second curve (with diamonds) shows the results for the decode and forward with encoding in which there is only one decoder at the destination. The signals received from the source and the relay are combined using MRC and are jointly
Figure 4.13: Comparison of the results for DEF and DNEF, Equal power.

decoded.

The third curve (with stars) shows the results for decode and forward with encoding scheme when both the source and the relay use the same (1296, 1/2) code. In this case, the destination is using two separate decoders to decode the signals received from the source and the relay. Then, it combines the soft outputs of the two decoders and decides about the final value of each bit.

The fourth curve (with x's) shows the simulation results for the case that the source and the relay use different codes. In this case, the code used by the source has a block length of 1944 bits and code rate 1/2. The relay uses a 1296, 3/4 code to
re-encode the signal. The decoding scheme at the destination is similar to the second case.

It can be seen that for the cases that the relay is closer to the destination \((d > 0.6)\), all four strategies perform similarly. But, for the cases that the relay is closer to the source \((d < 0.45)\), DNEF and DEF, which use a joint decoder, are less sensitive to the relay location and distance from the source. When a homogenous code is used for the SR and RD channels, we have the advantage of jointly decoding the codewords, which proves to be almost a factor of \(2\text{dB}\) beneficial for the case of \(d = 0.1\).

We should note that in this document, we have reported the simulation results for the \((1296, 1/2)\) and \((1296, 2/3)\), \((1944, 1/2)\), \((1944, 3/4)\) codes. All of these results hold for the other architecture-aware codes in the 802.11n family of LDPC codes and the extensions of them.

### 4.2 Distributed Partial Decoding

In both DNEF and DEF schemes, it is assumed that the relay performs full decoding of the LDPC codeword until it converges to a correct codeword. Depending on the signal to noise ratio of the channel, this can take up to a hundred iterations, which results in long processing delays at the relay.

Figure 4.14 shows percentage of bits that are in error after each decoding iteration for a 1296, 1/2 LDPC code in a two node system (direct link). It can be seen that if the SNR is low, and the number of bits in error are high, more iterations need to be
Figure 4.14: Percentage of bits in error vs. Number of decoding iterations for different SNR values, for an LDPC code of (1296,1/2). Results are for the direct link case.

performed to correct the error. By increasing the SNR, fewer iterations are required to reach a correct codeword.

For the cases that the relay is relatively close to the source, the signal is received with higher reliability at the relay than the destination. Therefore, the first few iterations correct most of the errors in the codeword.

Considering the DNEF scheme, in which the LDPC code used at the SR and RD channels are equal, there is no need to wait for the decoding at the relay to converge to a correct codeword before transmission. Therefore, the codeword may be partially decoded at the relay and transmitted to the destination for full decoding. By ‘partially decoding’, we mean decoding for a few iterations, which corrects the errors in both information bits and the parity bits, eliminating the final step of the decoding that
Figure 4.15: Distributed and partial decoding, effect of number of iterations at the relay and the destination on the performance, (1296,1/2) LDPC code.

verifies the convergence to the correct codeword. Also, instead of removing the parity bits and re-encoding the signal, we keep the corrected parity bits in the codeword, modulate the signal and re-transmit it to the destination. The destination receives two copies from the source and the relay and jointly decodes them. Our simulation results show that there is a major advantage in partially decoding the codeword at the relay and distributing the decoding process between the relay and the destination.
4.2.1 Performance of Distributed Partial Decoding

In this section, we present distributed partial decoding which has the benefits of reduced processing latency, reduced power consumption and smaller hardware complexity at the relay. This scheme can be used with any error correcting code with iterative decoding, such as low density parity check (LDPC) codes or turbo codes. In this thesis we present the results for LDPC codes but the same approach can be applied to other codes too.

Figures 4.15 and 4.16 show the effect of partial decoding with different numbers of iterations at the relay and at the destination (SRD) on the decoding performance. Compared with the basic single direct path, these figures show that if we have limited processing time/power, it is better to distribute the processing load between the relay and the destination. In the figures, different bit error curves are shown for the DNEF strategy with partial decoding at the relay. The number of decoding iterations at the relay and at the destination is shown in the parenthesis in the figure legends.

In figure 4.15, comparing the simulation results of the direct path with 20 iterations at the destination with the cooperative communication link with one iteration at the relay and 20 iterations at the destination (1, 20), shows that partially decoding with only one decoding iteration at the relay results in a major improvement in the decoding performance (1.8 dB gain in BER of $10^{-4}$), although the codeword at the relay still has many errors after one iteration.

Comparing the cases of (1,20) and (20,1) (Figure 4.15) and also (10,5), (5,10)
(Figure 4.16) shows that if we have limited number of iterations or, equivalently, limited latency tolerable in the system, it is better to perform a few iterations at the relay and more iterations at the destination. This result is very appealing in practice, since it translates into smaller processing delay and consumption of less power at the relay. This is especially appealing when the relays are battery operated devices that agree to relay the information of other users.

If we keep the number of iterations at the destination at a fixed number and increase the number of iterations at the relay (comparing the cases for (3,10), (5,10), (10,10)), we see that the first few iterations at the relay are very important, but the effect of the relay saturates as the number of iterations grows. This means that performing partial decoding with $3 \sim 5$ iterations at the relay and more iterations at the destination gives us most of the benefits of the cooperation with much less decoding latency and processing power.

4.2.2 Distributed Partial Decoding for Rayleigh Fading Channels

Until now, we have presented the benefits of distributed partial decoding on AWGN channels. In this section we show that the same benefits hold for other channels such as Rayleigh fading. We consider a block fading channel that is constant through transmission of one codeword block. The fading coefficient may change from one block to another.

Figure 4.17 illustrates the simulation results for distributed decoding in Rayleigh
Figure 4.16: Distributed and partial decoding, effect of number of iterations at the relay and the destination on the performance, (1296,1/2) LDPC code.

fading channel. In this figure, simulation results are shown for a system without a relay (i.e., direct link transmission) for comparison. The results for a maximum number of iterations equal to 20 and 50 are presented.

Comparing the cases of (1, 20) and (20, 1) shows that the iterations at the destination are more important. This result is intuitive because we need to do some work to be able to recover the data at the destination. Even if the relay recovers all the data, there are errors in the RD link that need to be detected and corrected at the destination.

Comparing the performance results of (10, 10) and (3, 10) shows the benefits of distributing the decoding and performing just a few iterations at the relay similar to
Figure 4.17: Distributed and partial decoding, Rayleigh fading channel, $(1296,1/2)$ LDPC code.

Figure 4.18: Zoomed version of figure 4.17.
the results for an AWGN channel. In figure 4.17, it may seem that the two curves overlap, but there is 0.05 dB difference between them as shown in the zoomed version in figure 4.18.

Discussion

One of the advantages of partial decoding is that we can use the same decoder architecture that was designed for two node communication in user cooperation scenarios. The only difference is that the parity bits are not discarded after the decoding. In partial decoding, fewer decoding iterations are performed on the codeword at the relay. Assuming that full decoding requires at least 10 decoding iterations to converge to a correct codeword, and assuming that only 3 iterations are performed during the partial decoding at the relay, there is a savings of more than 70% in the decoding process for the partial decoding compared to full decoding. This translates to 70% savings in decoding latency because there is a linear relationship between the decoding latency and the number of decoding iterations. Also, it leads to 70% reduction in processing power at the relay.

Since distributed partial decoding utilizes DNEF strategy, there is no need for the encoding hardware at the relay, which also leads to savings in terms of the power/area and latency.
Mathematical Description of Distributed Decoding

There are a few analytical methods in the literature for analysis of the behavior of LDPC codes, such as: density evolution, Gaussian approximation and EXIT charts.

The Gaussian approximation (GA) [61] shows the evolution of the LLRs of the messages from the variable nodes to the check nodes during the decoding process as decoding iterations tend to infinity. By using this method, one can find the threshold which is the infimum of all the noise variances such that the probability of error tends to infinity.

In this section, we use the Gaussian approximation tool to mathematically show that distributed decoding is advantageous. We analyze the changes in the density of the LLR messages as the iterations go forward and show that for $\sigma > \sigma^*$, the first few iterations have the most effect on the decoding results.

From Gaussian approximation, we can track the evolution of the messages by their means with the following equation:

$$m_u^{(l)} = \sum_{j=2}^{d_c} \rho_j \phi^{-1} \left( 1 - \left[ 1 - \sum_{i=2}^{d_t} \lambda_i \phi(m_u^{(0)} + (i - 1)m_u^{(l-1)}) \right]^{j-1} \right).$$

(4.8)

We are interested in the percentage of the changes in the value of $mu_0$ in each iteration. Specifically, we want to compute the value of $Omega(l)$.

$$\Omega(l) = \frac{(m_u^{(l)} - m_u^{(l-1)})}{m_u^{(l-1)}}$$

$$= \frac{m_u^{(l)}}{m_u^{(l-1)}} - 1$$

(4.9)
Figure 4.19: Percentage of change in the mean of LLR messages from variable nodes, for a noise level close to the threshold.

By substituting Eqn (4.8) into the above equation, we have:

\[
\begin{align*}
\Omega(l) &= \frac{m^{(l)} - m^{(l-1)}}{m^{(l-1)}} \\
&= \frac{\sum_{j=2}^{d_x} \rho_j \phi^{-1} \left( 1 - \left[ 1 - \sum_{i=2}^{d_t} \lambda_i \phi(m_{nu} + (i-1) m_{nu}^{(l-1)}) \right]^{j-1} \right)}{\sum_{j=2}^{d_x} \rho_j \phi^{-1} \left( 1 - \left[ 1 - \sum_{i=2}^{d_t} \lambda_i \phi(m_{nu} + (i-1) m_{nu}^{(l-2)}) \right]^{j-1} \right)} - 1
\end{align*}
\]

in which \( \phi(x) \) is equal to:

\[
\phi(x) = \begin{cases} 
1 - \frac{1}{\sqrt{4\pi x}} \int_{\mathbb{R}} \tanh \frac{u}{2} e^{-\frac{(u-x)^2}{4x}} du, & \text{if } x > 0; \\
1 & \text{if } x = 0.
\end{cases}
\]

If \( x < 10 \) then \( \phi(x) \) can be approximated with

\[
\phi(x) = e^{\alpha x^\gamma + \beta}
\]

in which \( \alpha = -0.4527, \beta = 0.0218, \) and \( \gamma = 0.86. \)
There is no closed form for equation (4.9), so we show its trend numerically in figure 4.19 for an LDPC code from Urbanke’s paper [61]. This figure shows that the mean value of the variable node messages has the most changes during the first few iterations. As the iterations continue, the change in the value of \( m_{u0} \) becomes smaller and smaller, which means that the decoding has almost converged to the correct value of \( m_u \).

Now, let's look at the probability of error \( P_e \) at the \( l^{th} \) iteration of LDPC decoding. [61] shows that the probability of error at \( l^{th} \) iteration can be written as:

\[
p_e(l) = \sum_{i=2}^{d_i} \lambda_i' Q \left( \sqrt{\frac{s + it_i}{2}} \right)
\]

(4.11)

in which

\[
\lambda_i' = \frac{\lambda_i / i}{\sum_{j=2}^{d_i} \lambda_j / j}.
\]

\[
s = m_{u0}
\]

\[
t_i = m_u^{(l)} = f(s, t_{l-1}).
\]

The probability of error decreases in each iteration. As an example, figure 4.20 shows the trend of \( P_e \). It can be seen that the decrease in \( P_e \) is maximum in the first iteration and it gets smaller and smaller in later iterations. This shows why distributing the decoding process between the relay and the destination is useful. At the relay, the first few iterations correct most of the errors and the rest is sent to the destination for correction.

To get a better intuition, we depict the values of \( \Delta P_e \), \( \Delta P_e / P_e \) in figures 4.21 and 4.22.
Figure 4.20: Probability of error vs. iteration number for a noise level close to the threshold.

4.2.3 A Protocol for Choosing the Decoding Load at the Relay and Destination

In this section we propose a procedure for determining the partial decoding load at the relay and at the destination for the distributed decoding.

- Decide on the value of $\eta$ which is the percentage of the errors required to be fixed at the relay. For example, $\eta = 80\%$.

- Plug the value of $\eta$, LDPC degree distribution parameters and channel parameters into equation (4.11) to calculate the value of $l$.

- $l$ is the number of iterations to be performed at the relay.
Figure 4.21: Change in the probability of error vs. iteration number for a noise level close to the threshold.

Figure 4.22: Percentage of change in the probability of error vs. iteration number for a noise level close to the threshold.
We should note that the higher the value of \( l \), the more processing is done at the relay with the expense of processing delay and power consumption at the relay.

We can perform the same procedure to determine the number of iterations at the destination. If the destination is a wireless base station, it can spend much more power on the processing because it is usually plugged into the wall power vs. the relay which is usually a battery-operated unit. Therefore, the number of iterations at the destination can be much larger than the corresponding number at the relay.

### 4.3 Conclusions

In this chapter, we showed that cooperative communications can still be beneficial even if we relax or change some of the unrealistic information theoretic assumptions about the relays. We presented the advantages of decode, no encode and forward relaying strategy at the relay over the original decode and forward scheme, and showed that DNEF performs as good as DEF with the advantage of no encoding latency and no encoding hardware.

We also proposed the distributed and partial decoding and showed that this scheme can drastically decrease the time and power spent at the relay while having negligible performance loss compared to the original decode and forward strategy. In addition, we showed that combining DNEF and distributed decoding can further decrease the processing time at the relay while maintaining the benefits of relaying.

Until now, we showed how to decrease the complexity of the cooperative system
with distributed partial decoding and enjoy major savings in terms of processing power and processing time. In the next Chapter, we talk about puncturing and present our contributions on how to decrease the transmission time and transmission power at the relay through puncturing. In addition, we present our results in reducing the complexity of incremental redundancy.
Chapter 5

Distributed Decoding with Puncturing

In this chapter, we present our results for integrate the distributed decoding technique in cooperative communications with puncturing at the relay. We show that in some scenarios there are major benefits in the integration. We show how to search for a good puncturing pattern for a family of quasi-cyclic low density parity check (LDPC) codes. We define a few metrics for comparing different puncturing patterns without exhaustive simulations.

In section 5.4, incremental redundancy is used, in which the relay transmits the punctured section of the codeword to the destination incrementally in case of a decoding failure. We show the benefits of reusing some of the information from different decoding processes.

5.1 Puncturing

Puncturing is the process of selecting and forwarding a portion of a packet to a receiver. Puncturing reduces the transmission time at the transmitter, since there are less bits in a punctured packet to be transmitted. Puncturing may be used to generate rate-compatible codes from a mother code. Therefore, a single decoder can decode all the punctured codes in a family. As a result, a decoder architecture may
be able to support a family of punctured codes without any extra overhead.

One of the main drawbacks of puncturing is the performance loss due to elimination of information-bearing bits from a packet or a codeword. However, the amount of performance loss can be minimized by carefully choosing the bits to be punctured.

In chapter 4, we analyzed cooperation when the relay transmits all the information that is available to it to the destination. In this Chapter, we investigate a few scenarios in which the relay punctures the codeword and sends a portion of the information available to it to the destination.

One of the main reasons for employing puncturing in cooperation is that cooperation may be seen as repetition coding. Because the destination receives two copies of the signal from the source and the relay. In some scenarios, this redundancy may be more than enough. By utilizing puncturing in cooperation, spectral efficiency of the system may be increased. In addition, puncturing can generate a family of codes with more flexibility in terms of code rate and block lengths. By removing the extra redundancy in a codeword while utilizing the cooperation through puncturing, one can increase the efficiency of the system.

In this section, we analyze structured puncturing patterns for quasi-cyclic LDPC (QC-LDPC) code constructions as shown in figure 3.1. The QC-LDPC codes are very suitable for efficient hardware design. The QC-LDPC codes are systematic codes, therefore, they have simpler encoding process. For detailed description of these codes we refer the reader to Chapters 2 and 3. Unlike turbo or convolutional codes, there
is no known optimal puncturing pattern in the literature that can be applied to all of the LDPC codes. Most of the puncturing patterns in the literature are fully random [113,116]. Random patterns are very costly in terms of storage and retrieval in hardware, because the location of all the punctured bits should be stored to be used in the reconstruction step. On the other hand, structured puncturing patterns have the advantage of very compact storage and easy reconstruction. For a detailed review of the papers in the literature, the reader is referred to the Section 2.3in Chapter 2.

Figure 5.1 illustrates a block diagram of a three node cooperative communication scheme that utilizes puncturing at the relay. A Source node transmits a codeword in broadcast mode to the relay and the destination. The source does not puncture the codeword before transmission. Therefore, it sends the complete codeword to both the relay and the destination. After some processing (e.g., full or partial decoding) on the codeword, the relay punctures some of the bits and sends a portion of the codeword to the destination. The destination optimally combines the two versions received from
Figure 5.2: Puncturing 0%, 25%, 50%, 75% of the bits and sending the rest of the bits.

the source and the relay. The destination decodes the combined codeword.

5.1.1 Puncturing Ratios

In this section, the relay punctures the received codeword after decoding and transmits part of the bits to the destination, instead of transmitting the complete codeword. We analyze the effects of puncturing on the performance of the cooperative communications. The following four cases are considered. In all of these cases, the source sends all the codeword to the destination. The relay sends only a portion of the bits to the destination.

- Case A: In this case, there is no puncturing at the relay; the complete codeword
is sent to the destination. We show this case as a baseline for comparison of the other cases.

- Case B: The relay punctures 25% of the codeword and transmits the rest to the destination. The packet consists of all the information bits plus 50% of the parity bits.

- Case C: The relay punctures 50% of the codeword and transmits the rest to the destination. The packet contains all the information bits and no parity bits.

- Case D: The relay punctures 75% of the codeword and transmits the rest to the destination. The packet contains 50% of the information bits and no parity bits.

In this chapter both source and relay utilize an LDPC code of block length 1296 bits and code rate 1/2 unless otherwise stated. Figure 5.2 shows the performance results for a cooperative communication system with different puncturing ratios as stated above. In all of these cases, the destination receives a complete version of the codeword from the source and the relay sends a portion of the codeword to the destination as noted in the legends of the figure 5.2. It can be seen that cooperation is very robust to puncturing at the relay and when the channel between the source and the destination is good, the relay may not need to send a complete codeword and may only send parts of the codeword. It should be noted that since the destination receives a complete codeword from the source, it is possible to puncture a large portion of
the codeword transmitted from the relay and still get an acceptable performance, even better than the direct path transmission without puncturing. As illustrated in the figure, there is a known performance loss due to puncturing and the larger the number of the punctured bits, the larger the performance loss.

Table 5.1 presents the effective rate and the puncturing percentage of the above four cases. Both the source and the relay have a coding rate of $R = 1/2$ in all cases.

\[
\text{Effective Rate} = \frac{\text{Total number of information bits transmitted}}{\text{Total number of bits transmitted}} = \frac{K}{(N_{SD} + N_{RD})}. \tag{5.1}
\]

<table>
<thead>
<tr>
<th>Case</th>
<th>Puncturing percentage at the relay</th>
<th>Portion of the codeword transmitted from the relay</th>
<th>Effective Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>complete codeword</td>
<td>1/4</td>
</tr>
<tr>
<td>B</td>
<td>25</td>
<td>3/4 of the codeword</td>
<td>1/3</td>
</tr>
<tr>
<td>C</td>
<td>50</td>
<td>1/2 of the codeword</td>
<td>2/7</td>
</tr>
<tr>
<td>D</td>
<td>75</td>
<td>1/4 of the codeword</td>
<td>2/5</td>
</tr>
</tbody>
</table>

In all of the simulations of this chapter it is assumed that the total power available at the source and at the relay is constant $P_{tot}$. Each of the source and the relay have half of the power available to them for transmission. Therefore, although the
relay sends a portion of the codeword to the destination, it has 1/2 of the total power available to it. For example, when the relay sends 1/2 of the codeword to the destination (Case C), the relay uses 1/2 of its available power. So the overall power used by the relay and the destination is:

\[ P_{CaseC} = P_{Source} + P_{Relay} = 1/2P_{tot} + 1/4P_{tot} = 3/4P_{tot}. \]

The above example shows that in order to have a fair comparison, the performance should be adjusted for the amount of the power that each of the systems use. The adjustment factor for cases B, C and D are \(-0.5799, -1.25, -2.0412\) dBs, respectively. In all of the figures in this section the power is adjusted unless otherwise is stated.

Cooperation can also be viewed as repetition coding which reduces the spectral efficiency. The above results show that in many cases we can decrease the amount of information sent from the relay, increase the spectral efficiency and still enjoy performance gain through cooperation.

5.1.2 Puncturing Patterns

In this section, we consider structured puncturing patterns that are uniformly distributed in a codeword. The patterns may be generated by keeping and removing blocks of \( S \) bits from the codeword as illustrated in figure 5.3. By varying the block size \( S \), six structured puncturing patterns are formed. Here we describe the puncturing patterns for an LDPC code of length 1296, rate 1/2, in which the sub-block sizes are \( S = 54 \).
Figure 5.3: Structured puncturing patterns
• Info only: In this puncturing pattern, the relay punctures all of the parity bits and transmits only the information bits to the destination. If this pattern is used, only one copy of the parity bits will be received from the source.

• Every other 54 bits: In this puncturing pattern, the relay punctures sub-blocks of 54 bits and transmits every other one of these blocks. Since the LDPC code that we are considering is block structured, this pattern is adjusted to the structure of the code.

• Every other 54 × 3 bits: In this puncturing pattern, the relay alternates between sending 3 blocks of 54 bits and puncturing 3 blocks and so forth.

• Every other 54 × 6 bits: For this puncturing pattern, the relay alternates between sending 6 blocks of 54 bits and puncturing 6 blocks.

• Parity only: In this puncturing pattern, the relay sends only the parity bits of a codeword to the destination. In this case, the destination receives one set of information bits and two sets of the parity bits from the source and the relay.

• Every other one: In this case, the relay punctures every other bit of the codeword. Therefore, half of the information bits and half of the parity bits are transmitted from the relay to the destination.

For a system using any of the above puncturing patterns, the destination receives a complete copy of the signal from the source, and a punctured copy from the relay.
and combines the two copies through MRC before decoding. It should be noted that these puncturing patterns are only a selection from the set of available patterns.

Figure 5.4 presents the bit error rate curves for comparing the above structured puncturing patterns. From the figure, it is obvious that in low SNRs, 'Info only' outperforms other cases and the sending every other bit outperforms cases 'Parity only' and 'Every other 54'. This shows that in low SNRs, in the event that we have limited resources for transmission and have to decide between transmitting the information bits or the parity bits, it is better to transmit the information bits. The parity bits can be sent in the second transmission if the destination is unable to decode the codeword (the same as the incremental redundancy scheme), but this is not always true. In high SNRs, the pattern 'Every other 54 x 3' is the best pattern.

The frame error rates (FER) for the above structured puncturing patterns are also shown in figure 5.4. FER is a better metric for comparing different puncturing patterns because it shows how many packets are in error after the puncturing/depuncturing/decoding. One of the main results in this figure is that puncturing both parity bits and information bits are beneficial and we should not limit ourselves to puncturing only the parity bits. It can be seen that the best puncturing pattern in terms of the FER is when blocks of 54 bits are punctured alternately.
Figure 5.4: Effect of utilizing structured puncturing patterns at the relay on the performance of a cooperative communication system.
5.2 A Metric for Comparing Puncturing Patterns

In this section we explore metrics for comparison between different puncturing patterns. Most of the papers in the literature about puncturing use exhaustive simulations to compare the performance of different puncturing patterns. In this section we look for a good metric by exploring the following metrics:

- Percentage of punctured branches
- Theoretical threshold using density evolution
- Average column(variable)/row(check) degrees
- Number of short cycles, Hamming distance

It is known that in LDPC codes, the variable nodes with higher degrees converge faster than the variable nodes with lower degrees. As a result, the higher degree nodes help the nodes with lower degrees to converge. If puncturing an LDPC codeword eliminates the variable nodes with high degrees, the resulting code converges slower and needs to perform more iterations to decode the codeword. For example, if we have a choice in the puncturing among two nodes with high and low degrees (e.g. 10 and 4), it is better to puncture the node with the lower degree (i.e., 4 in this example) and keep the node with the higher degree. Degree of a variable node refers to the number check nodes that are connected to the variable node or number of graph connections. Therefore, by puncturing a variable node, the corresponding graph connections or branches deleted.
5.2.1 Examples

In this section we present three examples for single link LDPC codes to give us insight in comparing the above metrics.

Example One

In this section we present an example LDPC code and puncture some portion of the variable nodes and compare the resulting codes in terms of their decoding thresholds.

Consider an irregular LDPC code with variable nodes of degrees \( i = 2, 6, 10, 20 \) having equal number of variable nodes for each degree. If all the codes symbols with degree \( j \) belong to a group denoted as \( G_j \) for \( 2 \leq j \leq d_i \), there will be equal elements in each group \( G_j \). Therefore, following the notation in [59], the degree distribution in terms of the nodes is as follows:

\[
\Lambda(x) = 0.25x + 0.25x^5 + 0.25x^9 + 0.25x^{19}, \quad (5.2)
\]
\[
\Delta(x) = x^{18}. \quad (5.3)
\]

The above node-based degree distributions can be converted to the following edge-based degree distributions:

\[
\lambda(x) = 0.05263x + 0.15789x^5 + 0.26316x^9 + 0.52632x^{19}, \quad (5.4)
\]
\[
\rho(x) = x^{18}. \quad (5.5)
\]

If the above code has a rate equal to 1/2, the noise threshold for will be equal to \( \sigma = 0.705 \).
To explore the effect of puncturing of low degree nodes and high degree nodes on the threshold, we puncture 50% or 100% of nodes with different degrees. Therefore, the puncturing pattern is as follows:

\[
\pi^{(0)}(x) = \pi_2^{(0)} x + \pi_6^{(0)} x^5 + \pi_{10}^{(0)} x^9 + \pi_{20}^{(0)} x^{19}
\]  

(5.6)

in which

a) \( \pi^{(0)}(x) = 0.5x \)

b) \( \pi^{(0)}(x) = x \)

c) \( \pi^{(0)}(x) = 0.5x^{19} \)

d) \( \pi^{(0)}(x) = x^{19} \)

In these punctured codes, either 12.5% or 25% of all the nodes are punctured. The resulting threshold values are reported in table 5.2. As we can see from the results, puncturing high degree nodes has more impact on the threshold of the code rather than low degree nodes. This is what we expected to see since the nodes with higher degrees converge faster and help the nodes with lower degrees to converge. Puncturing the same number of variable nodes from the highest degree nodes decreases the threshold much more than puncturing the same amount of low degree nodes.

Figure 5.5 shows the relationship between the puncturing proportion and the resulting threshold. In this figure, the threshold is calculated for the scenarios in which percentage of the punctured nodes from the different degree nodes are varied.
Table 5.2: Threshold values for different puncturing experiments for the LDPC code in Ex1, \( \lambda(x) = 0.05263x + 0.15789x^5 + 0.26316x^9 + 0.52632x^{19} \), \( \rho(x) = x^{18} \).

<table>
<thead>
<tr>
<th>Puncturing pattern</th>
<th>Punctured node degrees</th>
<th>Puncturing percentage</th>
<th>Total puncturing percentage</th>
<th>Threshold</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code (w/o punct)</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0.705</td>
<td>0.5</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>50%</td>
<td>12.5%</td>
<td>0.671</td>
<td>0.5714</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>100%</td>
<td>25%</td>
<td>0.633</td>
<td>0.6667</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>50%</td>
<td>12.5%</td>
<td>0.361</td>
<td>0.5714</td>
</tr>
<tr>
<td>d</td>
<td>20</td>
<td>100%</td>
<td>25%</td>
<td>0.119</td>
<td>0.6667</td>
</tr>
</tbody>
</table>

Figure 5.5: Puncturing percentage vs threshold for the LDPC code in Ex1, \( \lambda(x) = 0.05263x + 0.15789x^5 + 0.26316x^9 + 0.52632x^{19} \), \( \rho(x) = x^{18} \).
from 0% to 100%. Here, the three curves show the change in threshold vs percentage of the punctured nodes for the nodes with degrees 2, 10 and 20. It can be seen that puncturing the degree 20 nodes for higher than 51% does not have much effect on the threshold because the resulting code is already very bad and puncturing more nodes does not change it much. Also, change in the threshold is much smaller when the lowest degree nodes are punctured, which means that the puncturing has a smaller degrading effect on the performance.

The LDPC code used in the above experiment had a uniform degree distribution, it was not necessarily a good code, but was able suitable to separate the effect of puncturing different degree nodes on the threshold. A similar experiment was performed with a code from Urbanke's paper [61] which is optimized and has very good performance. The degree distributions for this code are as follows:

\[
\lambda(x) = 0.23403x + 0.21242x^2 + 0.14690x^5 + 0.10284x^6 + 0.30381x^{19}, \quad (5.7)
\]

\[
\rho(x) = 0.71875x^7 + 0.28125x^8. \quad (5.8)
\]

Table 5.3 reports the threshold values for this code. We can see a similar trend in this code as in the previous example.
Table 5.3: Threshold values for different puncturing experiments for the LDPC code in Urbanke’s paper, $\lambda(x) = 0.23403x + 0.21242x^2 + 0.14690x^5 + 0.10284x^6 + 0.30381x^{19}$, $\rho(x) = 0.71875x^7 + 0.28125x^8$.

<table>
<thead>
<tr>
<th>Puncturing pattern</th>
<th>Punctured node degrees</th>
<th>Puncturing percentage</th>
<th>Total puncturing percentage</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code (w/o punct)</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0.946</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>50%</td>
<td>12.5%</td>
<td>0.915</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>100%</td>
<td>25%</td>
<td>0.877</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>50%</td>
<td>12.5%</td>
<td>0.902</td>
</tr>
<tr>
<td>d</td>
<td>20</td>
<td>100%</td>
<td>25%</td>
<td>0.797</td>
</tr>
</tbody>
</table>
Example Two

In this example, the number of high degree nodes are 1/3 of the number of other degree nodes. Therefore, the degree distribution for an infinite-length LDPC code is as follows:

\[ \Lambda(x) = 0.3x + 0.3x^5 + 0.3x^9 + 0.1x^{19}, \]  \hspace{1cm} (5.9)

\[ \Delta(x) = x^8. \] \hspace{1cm} (5.10)

The above node-based degree distributions can be converted to the following edge-based degree distributions:

\[ \lambda(x) = 0.08108x + 0.24324x^5 + 0.40541x^9 + 0.27027x^{19}, \]  \hspace{1cm} (5.11)

\[ \rho(x) = x^8. \] \hspace{1cm} (5.12)

We repeated the above mentioned experiment with this code. Table 5.4 shows the threshold values for different puncturing ratios. Figure 5.6 shows the threshold vs. percentage of nodes of degree \( f \) and the threshold vs. total percentage of variable nodes punctured.
Figure 5.6: Puncturing percentage vs threshold for the LDPC code in Ex2, $\lambda(x) = 0.08108x + 0.24324x^5 + 0.40541x^9 + 0.27027x^{19}$, $\rho(x) = x^8$. 
Table 5.4: Threshold values for different puncturing experiments for the LDPC code in Ex2, $\lambda(x) = 0.08108x + 0.24324x^5 + 0.40541x^9 + 0.27027x^{19}$, $\rho(x) = x^8$.

<table>
<thead>
<tr>
<th>Puncturing pattern</th>
<th>Punctured node degrees</th>
<th>Puncturing percentage</th>
<th>Total punct. percentage</th>
<th>Threshold</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code (w/o punct)</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0.98</td>
<td>0.1778</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>50/3%</td>
<td>5%</td>
<td>0.96</td>
<td>0.1872</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100/3%</td>
<td>10%</td>
<td>0.95</td>
<td>0.1976</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>50%</td>
<td>15%</td>
<td>0.93</td>
<td>0.2092</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>100%</td>
<td>30%</td>
<td>0.88</td>
<td>0.2540</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>50%</td>
<td>5%</td>
<td>0.89</td>
<td>0.1872</td>
</tr>
<tr>
<td>d</td>
<td>20</td>
<td>100%</td>
<td>10%</td>
<td>0.78</td>
<td>0.1976</td>
</tr>
</tbody>
</table>
Example Three

In this example, the difference between the degrees of the low-degree nodes and high-degree nodes is smaller than the previous examples. The following degree distributions are considered for an infinite-length LDPC code:

\[ \Lambda(x) = 0.25x + 0.25x^3 + 0.25x^6 + 0.25x^9, \quad (5.13) \]
\[ \Delta(x) = x^6. \quad (5.14) \]

The edge-based degree distributions are as follows:

\[ \lambda(x) = 0.0870x + 0.1739x^3 + 0.3043x^6 + 0.4348x^9, \quad (5.15) \]
\[ \rho(x) = x^6. \quad (5.16) \]

We repeated a similar experiment with this code. The difference between this experiment and example 1 is that in this example the starting rate is lower and the code has higher threshold. Table 5.5 shows the threshold values for different puncturing patterns. Figure 5.7 shows the threshold vs. percentage of nodes of degree \( j \) punctured.
Figure 5.7: Puncturing percentage vs threshold for the LDPC code in Ex3, $\lambda(x) = 0.0870x + 0.1739x^3 + 0.3043x^6 + 0.4348x^9$, $\rho(x) = x^6$.

Table 5.5: Threshold values for different puncturing experiments with the LDPC code in Ex3, $\lambda(x) = 0.0870x + 0.1739x^3 + 0.3043x^6 + 0.4348x^9$, $\rho(x) = x^6$.

<table>
<thead>
<tr>
<th>Puncturing pattern</th>
<th>Punctured node degrees</th>
<th>Puncturing percentage</th>
<th>Total puncturing percentage</th>
<th>Threshold</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code (w/o punct)</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1.074</td>
<td>0.1786</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>50%</td>
<td>12.5%</td>
<td>1.023</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>100%</td>
<td>25%</td>
<td>0.968</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>50%</td>
<td>12.5%</td>
<td>0.909</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>20</td>
<td>100%</td>
<td>25%</td>
<td>0.677</td>
<td></td>
</tr>
</tbody>
</table>
5.2.2 Metrics

In this section, we investigate a few metrics for predicting the performance of structured puncturing patterns.

Percentage of Punctured Branches

Table 5.6 shows the number and the percentage of the punctured branches for different puncturing patterns for an LDPC code with a block length of 1296 bits and rate 1/2. In all of the cases reported in this table, half of the bits are punctured. It can be seen that if we choose the bits with lower degree to puncture, the number of total branches that are punctured are smaller and hence the resulting code performs better.

Table 5.6: Number and percentage of punctured branches for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2

<table>
<thead>
<tr>
<th>Puncturing Pattern</th>
<th>Total number of punctured branches</th>
<th>Percentage of punctured branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info only</td>
<td>1295</td>
<td>24.99%</td>
</tr>
<tr>
<td>Every other block of 54</td>
<td>2376</td>
<td>45.84%</td>
</tr>
<tr>
<td>Every other block of 54 × 3</td>
<td>2268</td>
<td>43.76%</td>
</tr>
<tr>
<td>Every other block of 54 × 6</td>
<td>1620</td>
<td>31.26%</td>
</tr>
<tr>
<td>Parity only</td>
<td>3888</td>
<td>75.01%</td>
</tr>
<tr>
<td>Every other one</td>
<td>2599</td>
<td>50.14%</td>
</tr>
</tbody>
</table>
Average Variable and Check Degrees

Two other metrics for comparing different puncturing patterns for a specific code can be the average row and column degrees of the resulting code after puncturing. Table 5.7 shows these metrics for an LDPC code with block length of 1296 bits and code rate 1/2.

Table 5.7: Average row and column degrees for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2

<table>
<thead>
<tr>
<th>Puncturing Pattern</th>
<th>Average Column Degree</th>
<th>Average Row Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Punctured</td>
<td>3.9992</td>
<td>7.9985</td>
</tr>
<tr>
<td>Info only</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Every other block of 54</td>
<td>2.1659</td>
<td>4.3318</td>
</tr>
<tr>
<td>Every other block of 54 × 3</td>
<td>2.2492</td>
<td>4.4985</td>
</tr>
<tr>
<td>Every other block of 54 × 6</td>
<td>2.7492</td>
<td>5.4985</td>
</tr>
<tr>
<td>Parity only</td>
<td>0.9992</td>
<td>1.9985</td>
</tr>
<tr>
<td>Every other one</td>
<td>1.992</td>
<td>3.9985</td>
</tr>
</tbody>
</table>

Number of Short Cycles

In this section we report number of short cycles [145] in the code after puncturing with the above puncturing patterns. The number of short cycles is a good estimate for the
Hamming distance of LDPC codes. The calculation of Hamming distance for LDPC codes is an NP-complete problem that cannot be done in polynomial time [146, 147] because of sparsity and dimensions of the parity check matrix.

Table 5.8 shows the number of short cycles of length four and six for the LDPC code 1296, 1/2. It can be seen that by puncturing some of the nodes, we are actually breaking some of the cycles and reducing the number of length-4 and length-6 cycles in the punctured code. Comparing the number of short cycles in each punctured code with the simulation results shown in section 5.3 indicates that this metric can describe the behavior of four out of the six patterns that we were testing. Number of short cycles can correctly predict the behavior (FER) of Info only, Every other 54, Every other 54 x 3, Every 54 x 6. The two patterns that do not follow the behavior of others are the Parity only and Every other one patterns. These two have the highest percentage of punctured branches, which seems to have more degrading effect than the number of short cycles. This can be intuitive. If the puncturing pattern cuts more than half of the connections in the graph, then the resulting punctured code can not perform well.

Threshold

We have calculated the threshold using the Gaussian approximation for an infinite-length degree distribution similar to a code in the IEEE 802.11n standard, with block length 1296 bits and code rate 1/2. It should be noted that the Gaussian
Table 5.8: Number of short cycles of length four and six for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2

<table>
<thead>
<tr>
<th>Puncturing Pattern</th>
<th># of length-4 cycles</th>
<th># of length-6 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Punctured</td>
<td>53</td>
<td>7236</td>
</tr>
<tr>
<td>Info only</td>
<td>0</td>
<td>6048</td>
</tr>
<tr>
<td>Every other block of 54</td>
<td>0</td>
<td>756</td>
</tr>
<tr>
<td>Every other block of 54 × 3</td>
<td>53</td>
<td>1188</td>
</tr>
<tr>
<td>Every other block of 54 × 6</td>
<td>53</td>
<td>4752</td>
</tr>
<tr>
<td>Parity only</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Every other one</td>
<td>0</td>
<td>864</td>
</tr>
</tbody>
</table>

approximation is accurate for computing the threshold for the infinite-length codes. So, we have assumed to have an infinite-length code with a degree distribution equal to the one in the above mentioned finite-length code. The threshold value for the original code is equal to $\sigma = 0.93573$. Puncturing distributions are computed for a few puncturing patterns. By using those, threshold values are calculated for the resulting punctured codes assuming infinite-length codes. Table 5.9 shows the values for the thresholds of the different puncturing patterns. As we can see, although gaussian approximation is a powerful tool for analyzing infinite-length code, it is not suitable to predict the performance of finite-length codes.

Looking at all the metrics explained above and in the examples, we can conclude
Table 5.9: The threshold values for different puncturing patterns in a LDPC code of length 1296 bits, rate 1/2, assuming infinite length code with the same degree distribution

<table>
<thead>
<tr>
<th>Puncturing Pattern</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code</td>
<td>0.93573</td>
</tr>
<tr>
<td>Info only</td>
<td>0.12360</td>
</tr>
<tr>
<td>Every other block of 54</td>
<td>0.12205</td>
</tr>
<tr>
<td>Parity only</td>
<td>0.12199</td>
</tr>
<tr>
<td>Every other one</td>
<td>0.12200</td>
</tr>
</tbody>
</table>

all of these metrics give us general insight of the code performance. A combination of a number of short cycles and percentage of punctured branches in the resulting punctured code can be used to predict the behavior of puncturing patterns at the relay. These metrics can be used to eliminate some of the worst puncturing patterns and focus on a smaller pool of possible patterns for exhaustive simulations.

5.3 Cooperative Communications with Puncturing at the Relay

The simulation results for a cooperative communication system utilizing puncturing at the relay are presented in this section.
5.3.1 Effect of Decoding Iterations on Puncturing

It is well known that puncturing degrades the performance. A viable question is whether or not increasing the number of decoding iterations at the relay and at the destination helps to recover completely from the errors introduced by puncturing.

The answer depends on the puncturing ratio and the puncturing pattern. Usually, if the number of punctured bits (i.e., puncturing ratio) is small, the decoder might be able to correctly guess the punctured bits. The higher the puncturing ratio, the more difficult to correct the punctured bits. The puncturing pattern also affects the result. If the puncturing ratio is large (for example if half of the bits in a codeword are punctured), it may not be possible to recover effect of puncturing completely, even by performing an infinite number of iterations.

Figure 5.8 demonstrates effect of decoding iterations at the relay and the destination on the performance of a cooperative communication system using punctured codeword at the relay. This figure shows the performance of a system without puncturing as a baseline. All other curves show the result for the case that half of the bits are punctured from the relay and only the information bits are sent to the destination.

Comparing the curves for (20, 20) and (100, 100) show that even if we increase the decoding iterations five fold, the bit error rate does not change significantly ($< 0.1 dB$). This means that some of the errors/punctured bits are not recoverable in the decoder even with more decoding iterations. It can be seen that the BER for the cases of (20, 20), (100, 20), (20, 100) and (100, 100) are similar, which means that increasing
Figure 5.8: Cooperative communications with puncturing at the relay, effect of decoding iterations on performance.
Figure 5.9: Cooperative communications with puncturing at the relay, effect of decoding iterations at the destination on performance.

the decoding iterations at the relay or the destination after a certain point (e.g., 20) does not provide much benefits.

Figure 5.9 shows the effect of the number of iterations at the destination on the performance for different puncturing ratios. It is intuitive to think that since we have punctured some of the bits, the destination needs to perform more decoding iterations to correct the errors caused by puncturing in addition to the errors caused by noise in transmission channel. The figure shows the results in four groups for the puncturing ratios of 0, 0.25, 0.5, 0.75. The legends in the figure show the percentage of the codeword that is sent, which has a one to one relation with puncturing ratios.
(refer to table 5.1). For each puncturing ratio, three performance curves are drawn showing a fixed number of iterations at the relay (i.e., 10) and variable iterations at the destination e.g., 3, 5, 10. Comparing the results for the case without puncturing and with puncturing indicates that the decoding iterations at the destination are very important in correcting the errors and do not saturate the result as fast as the case without puncturing. This result is intuitive, because there should be some extra effort at the destination to fix the effect of puncturing.

5.3.2 Distributed Decoding with Puncturing

In this section, we show the effect of puncturing on distributed and partial decoding. As we noted before, distributed decoding cuts the processing time and processing power at the relay. Puncturing cuts the transmission time/power at the relay. Now, we want to see if it is possible to take advantage of both distributed decoding and puncturing in the same system. The answer is yes, as we describe below.

Figure 5.10 depicts the bit error rate curves for a cooperative communication system utilizing puncturing at the relay. Several experiments with different numbers of iterations at the relay and at the destination are shown in the figure. Comparing the cases with 3, 10 iterations at the relay and 10 iterations at the destination (e.g. (3, 10), (10, 10)), verifies that distributed partial decoding can in fact be integrated with puncturing, without major performance loss. There is a small difference between the performance curves of partial decoding at the relay (e.g. 3 iterations) and full
decoding at the relay (e.g. 10 iterations). Therefore, combining distributed decoding and puncturing at the relay is possible. This is because if the relay fully decodes the codeword and then punctures some bits, it is actually wasting some energy correcting the errors on some of the bits that are eventually punctured.
Figure 5.10: Effect of distributed and partial decoding with puncturing on the performance.
5.3.3 Other Block Lengths and Code Rates

Until now, we have presented the results for an LDPC code with block length of 1296 bits and code rate 1/2, with different puncturing ratios. In this section, we present results for other block lengths and code rates and show that a similar trend exist in all of the codes in the 802.11n standard.

Tables 5.10 and 5.11 present number and percentage of punctured branches and average row and column degrees for different puncturing patterns for an LDPC code of length 1944 bits and rate 1/2. When comparing these two tables with tables 5.6 and 5.7, a similar pattern is seen for two different codes with block lengths 1296 and 1944 bits.

Figure 5.11 shows the performance of an (1944, 1/2). The performance of this code for different puncturing patterns follow the same trend or a (1296, 1/2) code.

It should be noted that in most of the simulations in this Chapter, code rate of 1/2 is assumed for the LDPC code under investigation. Other code rates (e.g., with higher rates) perform similarly with an offset in the SNR as presented and described in figure 4.9.
Table 5.10: Number and percentage of punctured branches for different puncturing patterns for an LDPC code of length 1944 bits, rate 1/2

<table>
<thead>
<tr>
<th>Puncturing Pattern</th>
<th>Total number of punctured branches</th>
<th>Percentage of punctured branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info only</td>
<td>1943</td>
<td>24.99%</td>
</tr>
<tr>
<td>Every other block of 54</td>
<td>3780</td>
<td>48.62%</td>
</tr>
<tr>
<td>Every other block of 54 × 3</td>
<td>3564</td>
<td>45.84%</td>
</tr>
<tr>
<td>Every other block of 54 × 6</td>
<td>2753</td>
<td>35.41%</td>
</tr>
<tr>
<td>Parity only</td>
<td>5832</td>
<td>75.01%</td>
</tr>
<tr>
<td>Every other one</td>
<td>3884</td>
<td>49.95%</td>
</tr>
</tbody>
</table>

Table 5.11: Average row and column degrees for different puncturing patterns in an LDPC code of length 1944 bits, rate 1/2

<table>
<thead>
<tr>
<th>Puncturing Pattern</th>
<th>Average Column Degree</th>
<th>Average Row Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Punctured</td>
<td>3.9995</td>
<td>7.9990</td>
</tr>
<tr>
<td>Info only</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Every other block of 54</td>
<td>2.0550</td>
<td>4.1101</td>
</tr>
<tr>
<td>Every other block of 54 × 3</td>
<td>2.1662</td>
<td>4.3323</td>
</tr>
<tr>
<td>Every other block of 54 × 6</td>
<td>2.5833</td>
<td>5.1667</td>
</tr>
<tr>
<td>Parity only</td>
<td>0.9995</td>
<td>1.9990</td>
</tr>
<tr>
<td>Every other one</td>
<td>2.0015</td>
<td>4.0031</td>
</tr>
</tbody>
</table>
Figure 5.11: Distributed and partial decoding, effect of puncturing on the performance.
5.4 Incremental Redundancy

Incremental redundancy refers to retransmission of the punctured bits incrementally in the event of failed decoding. Figure 5.12 shows a block diagram of a system using incremental redundancy. In this figure, a transmitter sends a punctured packet to a receiver at time $t = 1$. Upon receiving, the receiver tries to decode the packet at time $t = 2$. If the decoding fails, the receiver sends a request to the transmitter for more bits at time $t = 3$. Then, the transmitter sends the bits that were punctured and kept from the first transmission to the receiver at time $t = 4$. The receiver combines the two packets from two transmissions and starts decoding the combined packet at $t = 5$. Most of the papers in the literature that discuss incremental redundancy assume the second decoder starts the decoding from scratch. However, in this section we show that if we use some of the LLR values from the previous decoding, we can save some iterations of the second decoder without any loss in performance. To motivate our
Figure 5.13: Graphical representation of the errors in a codeword after each iteration of LDPC decoding.

Two observations are presented.

Figure 5.13 illustrates the errors that remain in an LDPC codeword after each decoding iteration. In this figure the blue dots show the bits that are in error and the white space means that the bits have been decoded correctly. Here we show the results for part of a codeword (200 bits). The errors in the first iteration are shown in the lower portion of the figure. As iterations continue, most of the errors are fixed. The remaining errors after 27 iterations are the errors that the decoder is not able to fix. From this figure, we can see that most of the bits, after they are fixed, remain correct and do not go into error in later iterations. The bits that fluctuate between being correct and in error have very small log likelihood ratio (LLR) values and are not reliable bits.

The relation between the absolute value of the LLRs and the errors associated with them is shown in Figure 5.14. This figure shows a histogram of the errors in a codeword after the first decoding iteration vs. LLR value of the bits in the codeword.
Figure 5.14: Histogram of number of bit-errors per each LLR value from a LDPC codeword.

In this experiment, 1000 codewords of length 1296 bits are transmitted at a low SNR. It is obvious that generally the smaller the LLR value, the less reliable the bit and the higher the probability of the bit being in error. Also, when the LLR values are greater than 10, number of errors in the corresponding bits tends to zero.

We present the result of a few experiments to show that we can extract some useful information from the first decoding process to use at the second decoding process. Since both of these decoding processes happen at the destination, it is useful to keep some of the values from the first set of decoding to use in the second set of decoding. Here is a list of the experiments:
• Combining the LLR values from the last iteration of the first decoding procedure with the new values received from the channel, and sending the combined signal to the decoder for the second decoding procedure.

• Checkpointing or keeping the LLR values from the first decoding procedure after a certain iteration and using it in the second decoding procedure.

• Using the LLRs that are larger than a certain threshold from the first decoding procedure.

• Clipping the LLR values from the first decoding procedure and using them in the second decoding procedure.

In this section our results for reusing the information from the first decoder in the second decoder while using incremental redundancy in the direct path case are presented. In the following simulations, an LDPC code with block length 1296 bits and code rate 1/2 is considered. The first 108 bits (i.e., the first two blocks) in the codeword are punctured which results in a puncturing ratio equal to 8.3%. When there is a decoding failure in the destination, it asks for more information from the source. The source sends all the bits that were previously punctured to the destination.

Frame error rates for the above scenarios in incremental redundancy are demonstrated in figures 5.16. The results suggest that for low SNRs, checkpointing at 5 iterations is very helpful and can save 15% of the decoding iterations of the second decoder. In low SNR regimes, clipping is not useful because most of the LLR values
Figure 5.15: Performance of direct path communication with incremental redundancy.

are small. At high SNR values, there is 5% savings if the LLRs from the previous decoder is clipped before usage at the second decoder.

Because the incremental redundancy is usually used in low SNR regimes when the quality of the signal is bad with a failed decoding event, our proposed scheme shows 15% savings in terms of processing power and time at the second decoder.

5.4.1 Hardware Overhead

In this section, we estimate the hardware overhead for the proposed scheme when using incremental redundancy. For our estimates, we use the information about
Figure 5.16: Frame error rate, performance of direct path communication with incremental redundancy.

Chartered 0.13\(\mu\) CMOS technology [148]. In this technology, the size of a two-input NAND gate is 4.5\(mm^2\) and the cell area for a single-port RAM memory is 3.61\(mm^2\).

For an LDPC code with a block length of \(N\) bits, the check-pointing requires \(N\) memory units. Since we are interested in the LLR values smaller than 10, each of these memory units can be described with 4 bits. This sums up to 4\(N\) memory bits. The comparison of the LLR values with a constant requires comparators. Since this process can be done offline, we do not need to have \(N\) comparators. Instead, we can have a semi-parallel design and reuse the comparator hardware for a group of LLR values.
Now, as an example, we calculate the hardware requirements for the incremental redundancy unit considering the parameters of our flexible LDPC decoder architecture described in Chapter 3. In this architecture, the maximum supported block length is 1944 bits, which refers to a quasi-cyclic LDPC code with subblock size of 81 bits. The memory required for our proposed scheme is 1944 × 4 = 7.7K bits = 0.97K Bytes.

If we consider a semi-parallel design with a folding factor of 24 (because of the design of the code), 81 comparators should be present in the hardware. Assuming each comparator needs a full adder and a multiplexer, this translates to 1620 gates. Using the sizing information from 0.13 micron CMOS technology, we need 0.00729mm² for logic and 0.028mm² for memory. Considering that the routing will add an extra 30% overhead, the total area overhead required in the architecture to support reusing of the information from the first decoding procedure in the second decoding procedure in a system utilizing incremental redundancy is 0.046mm². The area for a flexible LDPC decoder architecture described in Chapter 3 was 3.6854mm². By comparing these two numbers, the overhead for the reusing unit for incremental redundancy is 1.2% which results in 15% saving in processing time and power.

5.4.2 Discussion

It should be noted that the above savings are valid in the direct path scenario when the puncturing ratio is low. If the puncturing ratio is high, the LLR values after a few iterations are not reliable because existence of many errors (i.e., punctured bits)
has decreased the reliability of the messages that are passed among the nodes in the decoding process. In the cases with high puncturing ratios, it is better to start the second decoding procedure from scratch.

For cooperative communications with puncturing described in this section, in order to increase the spectral efficiency of the system through puncturing at the relay, high puncturing ratios should be used at the relay. Therefore, reusing the information in incremental redundancy does not result in savings.

5.5 Conclusions

In this chapter we presented structured puncturing patterns for quasi-cyclic LDPC codes and showed their behavior through simulations. We also presented puncturing metrics for finding a good puncturing pattern. Also showed the benefits of combining the distributed decoding with puncturing at the relay. In addition, we demonstrated the incremental redundancy and the benefits of retrieving some information from the previous decoding.

The next chapter will present a conclusion for the thesis and propose a few ideas for the extensions of this thesis in future.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we analyzed the problem of cooperative communications in wireless communication systems and offered solutions that enable a major reduction in complexity of these systems. Cooperative communications has shown to be very promising for the next generation of wireless devices. There exist many challenges ahead of us to be able to use cooperation in practice. These challenges include architectural complexity of systems supporting cooperation, resources used through cooperation and the time spent on cooperation. Although cooperation is promising, it is not feasible with the current algorithms and hardware. The current information-theoretic analysis of cooperation shows its benefits but with some unrealistic and often contradictory assumptions such as zero processing delay at the relay while performing full decoding on infinitely long codes. In this thesis, we showed how to change the algorithms to keep the benefits of cooperation while substantially reducing the required resources. Our goal was to reduce the gap between the theory and practice.

In this thesis, we set up a framework for system design and showed adaptability of our proposed techniques to different scenarios. This work was focused on the decode
and forward relay communication scheme because of its superior performance but with the challenge of higher complexity compared to other types of cooperation such as amplify and forward, and estimate and forward. Since decode and forward considers a full decoding/reencoding process at the relay, it is very resource-hungry. The decoding process is an enormous bottleneck in relay environments since it requires a substantial amount of power and takes a considerable amount of time. Our results show how to significantly reduce the cost of the system while maintaining good performance. The simulation results presented in Chapter 4 verify that all of our results hold.

Instead of having fixed relays that can be seen as smaller base stations, we would like to use the existing mobile units or handsets as a relay. These devices have limitations in terms of their size and often operate on batteries. Hence, they do not have large amounts of power and processor resources to offer for cooperation. We proposed distributed partial decoding in Chapter 4 which reduces the processing power and time at the relay by 70% while having minor performance degradation compared to the original decode and forward decoding. We also showed the trade-offs in selecting the relays in terms of their position and the ratio of power spent at the source and the relay.

In Chapter 5, we showed how to increase the spectral efficiency of the cooperative system by utilizing punctured LDPC codes. Parameters for designing good puncturing patterns for LDPC codes suitable for cooperation were identified. We also offered new algorithms for cooperation using punctured LDPC codes with incremental re-
Figure 6.1: Three node cooperative communications with LDPC coding.

dundancy. It was shown that by reusing the information from the first decoder, we can get up to 15% savings with less than 1.2% increase in hardware.

Figure 6.1 illustrates a block diagram of a cooperative communications system including a source, a relay and a destination, in which LDPC is the utilized channel coding. Figure 6.2 illustrates a similar block diagram in which the contributions of this thesis are marked. We presented techniques such as distributed partial decoding [6], decode and forward without encoding (DNEF) [7], and puncturing with incremental redundancy to decrease the processing and transmission time and power at the relay.

Architectural complexity of cooperation has been analyzed throughout the thesis. We showed how to reduce the hardware required at the relay by using the decode, no encode and forward strategy in addition to distributed partial decoding. We also explained the design methodology and provided the details of implementing a flexible
Figure 6.2: Summary of the thesis contributions on a three node cooperative communications block diagram.

LDPC decoder architecture [8,9] in Chapter 3. This decoder can be used as a stand-alone decoder or be used in cooperative environments.

All said, we can conclude that we have achieved all the goals of this thesis by considerable reduction in the complexity of the systems in terms of the area, power consumption and time. Our results are not limited to wireless environments and cell phones. The techniques offered in the thesis can also be applied to sensor networks and other emerging applications that have stringent power and processing requirements on the nodes.

6.2 Future Work

Although we analyzed many aspects of cooperative communication systems, there are still many open questions that can be considered as a continuation of this thesis.
Throughout the thesis, we assumed that the source and the destination know where the best relay is. There is a wealth of literature on relay selection strategies [1, 141, 142] and there is significant ongoing research on this problem. We chose to focus on the problems faced after the best relay is selected. Our results can readily be combined with relay selection strategies to show a complete multi-level system solution of the problem.

Another assumption throughout the thesis is that the source, relay and the destination lie in a direct line. One can assume more general topologies and extend our results to these environments.

Our results shed some light on the complexity reduction techniques that can be used in many other applications such as wireless sensor networks. These sensors usually have very stringent power requirements and cooperate with other nodes to transmit their data to a gateway sensor node.
Bibliography


2008.


[133] H.K. Yoon, H.G. Joo, and D.J. Shin. Construction of Rate-Compatible Block-Type LDPC Codes using Extending and Puncturing. In *IEEE 18th In-


[148] 0.13-micron CMOS ASIC TC280 Family.  