A MODIFIED BELIEF-PROPAGATION DECODER FOR THE PARALLEL DECODING OF PRODUCT CODES

Benjamin Martin Sim

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, in fulfillment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 2013

Contents

DECLARATION

I declare that this dissertation is my own unaided work. It is being submitted to the Degree of Master of Science to the University of the Witwatersrand, Johannesburg. it has not been submitted before for any degree or examination at any other University

.....
(Signature of Candidate)

..... day of ..... year .....

ABSTRACT

In this dissertation a modification to the belief-propagation algorithm is presented. The algorithm modifies the belief-propagation algorithm to allow for the parallel decoding of product codes. The algorithm leverages the fact that each component code in the product code can be independently decoded because the codewords are encoded by independent and identically distributed (i.i.d.) processes. The algorithm maximises the parellelisation by decoding all the component codes in each dimension in parallel. In order to facilitate this process we developed new additional stages which are added to the belief-propagation algorithm: the codeword reliability estimation, the belief-aggregation and the exit test stages. The parallel product code decoder offers a 0.2 dB worsening of the decoding BER performance when compared to the best serial decoder. However, the parallel belief-propagation decoder offers a 7.26 time *speedup* on an eight-core processor, which is 0.91 of the theoretical maximum of eight for an eight-core processor.

ACKNOWLEDGEMENTS

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

The author acknowledges the assistance of Jade More for assistance in proof reading the work.

List of Figures

List of Tables

List of Theorems

List of Algorithms

LIST OF SYMBOLS

a	-	Amplitude of a BPSK modulator
α	-	Percentage of an algorithm that is inherently serial
β	-	Weighting matrix
c_n	-	A binary symbol vector
d_{min}	-	Minimum distance of a code
e	-	Channel error vector (Gaussian Channel)
e	-	Number of erasures correctable by a code
η	-	Non-zero locations in H matrix
H	-	Parity-Check Matrix
Λ	-	Redundant processing
λ	-	Processing product
λ_{min}	-	Minimum λ for a decoding
m	-	Length of parity of a code
n	-	Encoded length of a code
n_h	-	Encoded length of the horizontal component code
n_v	-	Encoded length of the vertical component code
ω	-	The number of inner loops (belief-propagation iterations) performed
Ω	-	The number of outer loops performed
p	-	Number of processors in a parallel system
P	-	A product codeword
P^T	-	A transposed product codeword (duplicate in parallel decoder)
P'	-	Output product codeword (from parallel decoder)
ϕ	-	Weighting Vector
r_n	-	Received signal vector
ρ	-	Normalisation factor for the weighting matrix
S	-	*Speedup* of a parallel algorithm

σ^2	-	Noise variance
t_n	-	Transmitted signal vector
T_S	-	Serial processing time
T_P	-	Parallel processing time
t	-	Number of errors correctable by a code

ACRONYMS

ABP	-	Adaptive Belief Propagation
ARQ	-	Automatic Repeat reQuest
AWGN	-	Additive White Gaussian Noise
BCJR	-	Bahl-Cocke-Jelinek-Raviv
BCH	-	Bose-Chaudhuri-Hocquenghem
BEC	-	Binary Erasure Channel
BER	-	Bit Error Rate
BP	-	Belief-Propagation
BPSK	-	Binary Phase Shift Keying
CUDA	-	Compute Unified Device Architecture
dB	-	decibels
ECC	-	Error Control Coding
FEC	-	Forward Error Correction
GPU	-	Graphics Processing Unit
HTTP	-	Hyper Text Transfer Protocol
i.i.d.	-	independent and identically distributed
KLD	-	Kullback-Liebler Distance
LDPC	-	Low-Density Parity-Check
M-ABP	-	Modified Adaptive Belief Propagation
MAP	-	Maximum *A Posteriori*
ML	-	Maximum Likelihood
PC	-	Product Code
PLC	-	Power Line Communications

RS	- Reed-Solomon
RAM	- Random Access Memory
SNR	- Signal to Noise Ratio
SPA	- Sum-Product Algorithm
TCP	- Transmission Control Protocol

1 INTRODUCTION

In modern day telecommunications, the importance of transmitting information across a noisy channel is paramount to the success of any communication system. In digital communications, error free transmissions can be achieved through two different processes: Automatic Repeat reQuest (ARQ) or Forward Error Correction (FEC). ARQ techniques request the retransmission of information that has been corrupted during propagation through a channel. An ARQ technique will request the retransmission of the information until the data is correctly transmitted across the channel [1]. FEC, or channel coding, is a mathematical process of encoding the information at the transmitter prior to transmission, to facilitate the recovery of the original information at the receiver in spite of the presence of errors [2–4]. The limitation of an ARQ system is the unlikelihood of a successful retransmit in a channel with sufficient noise. Therefore, the transmission throughput is exponentially reduced as each packet has to be re-sent multiple times [2]. An FEC system can only recover from a finite number of errors; if the error threshold is exceeded, the errors cannot be corrected and a retransmit of that packet is required. To achieve successful transmission across the channel would require more parity to be added to the original information (through the use of a more powerful coding technique), which degrades the throughput of message data, thereby rendering the more powerful FEC techniques impractical. Thus, most modern systems are hybridised, utilising both techniques. It is evident that many communications systems are dependent upon both these concepts. As a result of this, any work within either field could provide significant advantage to a communication system. This dissertation presents an investigation into the topic of FEC.

The product code was first introduced by Elias in 1954 [5]. Rather than the typical structure of a binary string that is encoded, the product code is a binary block containing many binary strings that are independently encoded. The product code would later be classified as the first ever coding technique in the powerful turbo code group of coding techniques [6]. Product codes are considered turbo codes as information can be shared between the many component codewords during decoding. This sharing of information results in the decoding being an iterative process and offers decoding time performance improvements. Turbo codes were some of the first codes to approach the Shannon limit [7]. Due to this good performance, turbo codes are used in some of the harshest channels [8, 9]. Although, the term turbo does not, in fact, refer to the code's decoding Bit Error Rate (BER) performance, but rather

refers to the relatively short decoding time required for these codes. The decoding algorithms for turbo codes show good decoding performance for long codes [6, 7]. One limitation of the product code, due to its high number of component codes, is the costliness of decoding with lengthy decoding times. This problem is amplified in the case of the codes presented in this work, where the component codes used are Low-Density Parity-Check (LDPC) codes. The resulting LDPC product codes would have such a long decoding time that they become completely impractical in real world implementations. Conversely, LDPC product codes could offer impressive decoding BER performance making them ideal for very harsh channels. In this work the decoding time performance of LDPC product codes is investigated and a design for a parallel belief-propagation decoder for the decoding of LDPC codes is presented [10].

The current silicon based processor architectures are reaching the limit of their usable lifetimes [11]. As a consequence, the doubling of the number of transistors on a processor every eighteen months, as predicted by Moore's Law, is bound to reach its limit in the near future [12, 13]. One solution to this problem is the parallelisation of the current serial algorithms. Parallel algorithms offer massive performance improvements as the performance can theoretically scale linearly with the number of processors used [14]. These improvements no longer require large processing potential per processor, but rather a large pool of processing available across a cluster of processors. The structure of the product code, as a combination of independently encoded codewords, makes it an ideal platform for parallel decoding as each component codeword can be individually decoded. Combining this with the use of LDPC codes, which are typically long codes, the algorithm could leverage multiple parallel processors to dramatically improve the decoding performance.

1.1 Research Aims

This research aims to develop a parallel belief-propagation decoder for the decoding of LDPC product codes. The work will focus on the decoding time performance benefits offered rather than the decoding BER performance, although an algorithm which offers a weak decoding BER performance will still be undesirable. The work would need to offer the decoding time performance benefits at a decoding BER performance comparable to other similar techniques.

1.2 Research Outputs

A design of a modified belief-propagation algorithm for the decoding of LDPC product codes is presented in this work. The decoding time of LDPC product codes is significantly decreased by parallelising the decoding process, which is a topic that has not had significant investigation in literature. The work extends the basic belief-propagation algorithm by including three new additional processes which facilitate

the parallel decoding: the codeword reliability estimation, the belief-aggregation and the exit test stages. A complete analysis of the decoder is given with the performance benefits of all the permutations of the new extensions to the belief-propagation algorithm being detailed.

More practically, this work could be used to improve the transmission rate through harsh channels, such as the wireless or Power Line Communications (PLC) channels, due to the powerful BER decoding performance offered by an LDPC product code. A more forward thinking application would be in the transmission of large data packets from a smartphone. Typically wireless transmission requires retransmissions as large sections of a transmission may be lost to fades in the transmission, which is expected on a Rayleigh Fading channel [7]. If a single transmission could recover from significant corruption the need for retransmission would be significantly reduced and as a result the throughput would increase. However, a smartphone's processor is not capable of the computational power required for the encoding or decoding of such large product codes. One solution would be an *ad hoc* smartphone cluster to pool the processing power of the handsets in the immediate vicinity [15]. The algorithm that is presented in this paper would allow the decoding process to be spread across such a cluster to achieve faster decoding and, consequently, more reliable communication.

1.3 Dissertation Outline

The remainder of the dissertation is organised as follows. Chapter 2 presents the background and preliminary information needed to understand the work. In Chapter 3 and 4 two of the algorithms designed to parallelise the decoding of the product code will be presented and discussed. The algorithm in Chapter 3 is fundamentally flawed and as such never made it to testing, however, the process of the design illustrates the concepts and leads onto the final solution presented in Chapter 4. The results achieved by the second algorithm are presented in Chapter 5. The results presented are also analysed in Chapter 5 to ascertain whether the algorithm meets the required aims outlined in Section 1.1. Finally, some concluding assessments are made and future research paths outlined in Chapter 6.

2 LITERATURE REVIEW

Throughout the dissertation an understanding of certain concepts used in the work is required. In this chapter the concepts or fields which lead to or contribute to the development of the work are introduced. The main focus of this chapter will be a discussion on product codes, the belief-propagation algorithm and parallel performance. These three concepts form the basis upon which this work is built.

2.1 Product Codes

The product code was first described by Peter Elias in 1954 [5]. Elias described a coding system which encoded a block of data by concatenating the coding process of multiple simpler codes into one more powerful code. The resulting encoded block could theoretically be a hypercube of infinite dimensions. The product code was the first implementation of what would later be known as a turbo code. Turbo codes only became widely known and used following the work by Berrou, Glavieux and Thitimajshima [6], when they presented their work on concatenated convolutional codes which approached the Shannon limit. The main feature of a turbo code is the combination of two or more codes to achieve better performance. During decoding, the information is shared between decoders and allows the decoding to occur more rapidly. It is clear that this philosophy can be applied to a product code as well. As such the product code is a special kind of turbo code.

In this work all the product codes discussed are assumed to be binary two dimensional product codes, unless otherwise stated. This significantly simplifies the discussions down to an easily understandable level. Although the work only looks at two dimensional product codes, the decoder presented can easily be expanded to encompass any number of dimensions. The codewords which make up the product codeword are known as the component codewords.

2.1.1 Encoding

A product code differs from traditional codes as it encodes a block of data rather than a string. The data block can be viewed as a collection of messages to be encoded. Each row and column of the data block is thus encoded as an individual codeword, through separate independent and identically distributed (i.i.d.) encoding processes. Firstly, each of the row messages is encoded creating a horizontal parity section. Then each column, including the parity columns from the row encoding, is encoded creating a vertical parity section. The horizontal parity that was encoded vertically

creates a final parity on parity section. This overall structure, shown in Figure 2.1, presents the general structure for a two dimensional product code.

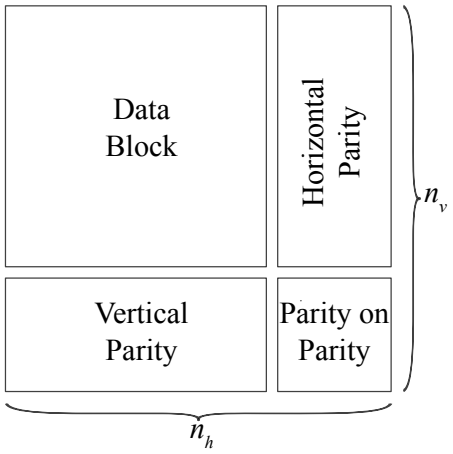

Figure 2.1: The structure of a two dimensional product code.

Product codes are typically constructed using a single component code, which simplifies their use, although it is possible to have two different codes. The length of a code is denoted by n, thus, the length of the horizontal and vertical component codes, if using two different coding techniques, will be given by n_h and n_v respectively. Both the rate and the minimum Hamming distance of the overall product code is a product of the rates and minimum Hamming distances of all the component codes [7]. The structure can be extended to encompass any number of dimensions; see Figure 2.2 which shows a three dimensional product code. However, it becomes challenging to visualise the structure beyond three dimensions.

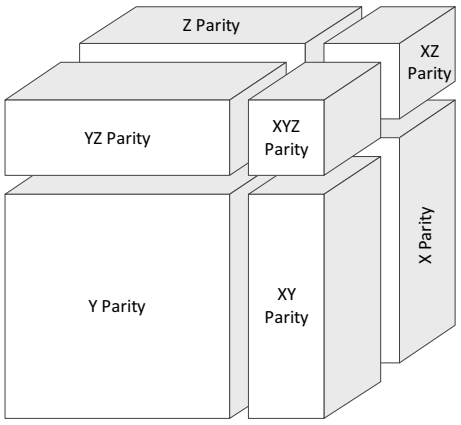

Figure 2.2: The structure of a three dimensional product code.

Initially, all work around product codes was focused around the construction of these codes. In Elias's paper he suggested that these codes could be constructed by codes such as Hamming, Golay or Reed-Muller codes [5]. Elias stipulated that any codes used would need to be in systematic form. A non-systematic encoding for the

vertical encoding would disperse the information for the horizontal encodings. This dispersal would introduce errors into the horizontal encodings. The cyclic product code, introduced by Burton and Weldon [16], was the only major improvement for the product code construction. The addition of cyclic component codes to the product code allowed the product code to be designed to meet specific channel requirements, due to the algebraic construction of cyclic codes. However, the product code can use any systematic code as its component code and thus leverage the benefits offered by that code's construction in the product code. The main feature to be taken from the encoding of product codes is that although different component coding techniques can be used, the basic hypercube structure of the product code has been maintained [17].

2.1.2 Hard-Decision Decoding

As the field of product code construction has remained mostly stagnant since its inception, the majority of work in the field has been surrounding the decoding of product codes. In his paper, Elias mentions that the decoding complexity of a product code is only as great as that of a single component codeword, albeit iterated through each component codeword [5]. The first hard-decision algorithm which focused on the decoding of product codes was presented by Reddy, later assisted by Robinson [18, 19]. The algorithm proposed by Reddy and Robinson was a hard-decision decoder which used an estimation of the BER in a component codeword, to apply a weighting factor indicating the codeword's reliability. Following the decoding of both the row and column codewords, it is necessary to recombine all the individual component codewords into a unified product codeword. Consequently, during the recombination the weighting factors are used to indicate the unreliable or less reliable component codewords, so that their impact on the final product codeword can be mitigated.

2.1.3 Soft-Decision Decoding

Soft-decision decoding can typically offer a three decibels (dB) Signal to Noise Ratio (SNR) performance improvement when compared to hard-decision decoding; see for instance [7]. Due to the performance benefits offered by soft-decision decoding, most recent work has been focused on this topic. All the soft-decision decoding algorithms for product codes are loosely derived from three techniques; Belief-Propagation (BP), Chase and trellis based decoding. All the soft-decision decoders, often unintentionally, make use of the Berrou *et al*'s turbo decoding principle [6]. Information from a single decoding iteration of one decoder is fed into the other decoders as extrinsic information.

Belief-Propagation Decoders

The belief-propagation algorithm, also referred to as the Sum-Product Algorithm (SPA), was first used in telecommunications by Gallager's iterative decoder for LDPC codes. However, the decoder presented by Gallager was a special case of the generalised belief-propagation algorithm. The generalised concept of belief-propagation was introduced by Judea Pearl in his seminal paper [10]. Belief-propagation is a form of Maximum *A Posteriori* (MAP) decoding that is used for the decoding of linear codes. The belief-propagation is a statistical inference algorithm for Baysian networks. The Tanner graph forms the Bayesian network upon which the belief-propagation operates. One weakness of belief-propagation is its poor performance when decoding codes with high-density parity-check matrices due to the short girth cycles within the graph. As a result of this the belief-propagation algorithm is often used in the decoding of LDPC codes, due to the sparsity of their parity-check matrices. Subsequently, Jiang introduced Adaptive Belief Propagation (ABP) which modified the belief-propagation algorithm for Reed-Solomon (RS) codes by adding a sparsification step before a belief-propagation iteration [20]. However, the ABP added a significant amount of complexity to the decoding process. Jégo and Gross modified the ABP further for specific use with product codes by modifying the ABP algorithm [21]. This new Modified Adaptive Belief Propagation (M-ABP) algorithm reduced the complexity of the original ABP making it more applicable to product code decoding. Jégo's M-ABP was shown to be successful for BCH codes.

Chase Decoders

The Chase decoding algorithm was first introduced by Chase in 1972 [22]. Unlike the belief-propagation algorithm, Chase decoding is a near Maximum Likelihood (ML) decoder. The Chase decoder selects a list of most possible codewords based on reliability information received from the demodulator. Then from that list an exhaustive search is used to determine the best codeword. The order of the exhaustive search is significantly decreased by first selecting a sub set of codewords rather than searching all possible codewords. This sub-set will increase with the minimum Hamming distance of the codeword. Hence, codewords with high minimum Hamming distances using a Chase decoder could struggle with impractical decoding times. In 1996, Pyndiah *et al* presented their modifications to the Chase algorithm, the Chase-Pyndiah algorithm [23]. Later, in 1998, Pyndiah presented a more formalised paper with some improvements from the original [17]. Pyndiah's algorithm incorporated a 'reliability of decision' step, between row and column decoding, to determine the confidence of a given codeword, determined by the Chase decoder. This reliability factor is then used when the algorithm recombines all the independent codewords from the row and column decodings. The poorly weighted codewords are discounted, thereby increasing the overall reliability of the product code.

Trellis-based Decoders

Some of the first work into the soft-decision decoding of product codes was presented by Lodge *et al* [24] and Hagenauer *et al.* [25]. Lodge presents what he termed a "Separable MAP filter" technique of decoding product codes. Lodge's work can be viewed, very simplistically, as separating each dimension into two distinct events which can be processed independently. The limitations of both Lodge's and Hagenauer's works lie in their use of trellis based structures for the decoding. Subsequently, the codes will be limited to small product codes as the number of states present in the trellis will increase exponentially with the size of the product code [17]. Trellis based decoders can be decoded through both MAP and ML decoding using algorithms derived from either the Viterbi [26] or Bahl-Cocke-Jelinek-Raviv (BCJR) [27] algorithms.

For the decoding of LDPC product codes the only option is the belief-propagation algorithm. Although the Chase-Pyndiah algorithm has been specifically modified to suit the product code, it becomes impractical for LDPC product codes. The Chase algorithm performs a search of a reduced set of positions within the codeword [7,22]. The search in the Chase algorithm is exhaustive over the reduced set and therefore has a factorial expansion based on the length of the reduced set. The number of positions in the set is based on the minimum Hamming distance of the code. Thus, for codes with a relatively short minimum Hamming distance the search can be completed within a reasonable time frame. However, LDPC codes have a very large minimum Hamming distance and as a result the set to be searched by the Chase decoder is far too large to perform an exhaustive search. Furthermore, the complexity of trellis-based decoder is exponentially related to the length of the code. For codes with the length of an LDPC code, trellis-based decoders would, again, be completely impractical [17], whereas the belief-propagation algorithm scales linearly with the length of the code and thus offers the best avenue for the decoding of LDPC product codes.

2.2 Belief-Propagation

Belief-propagation is a statistical inference algorithm for Bayesian networks created by Judea Pearl [10]. Belief-propagation does as its name suggests, and propagates the conditional beliefs of each node through the network until a state is reached where all the conditionals within the network are in agreement. The belief-propagation algorithm is used on Bayesian networks which have a bipartite construction. The marginals calculated for each node in one set of nodes in the bipartite graph are propagated, or passed as a message, to the other set of nodes. This is why the algorithm is often called the message passing algorithm. The marginals from the other set of nodes are received as a *priori* to be combined with all the conditionals from the current set to calculate a new marginal for each node. This process is

iterated until all the marginals for the entire network agree. This algorithm can be used for the decoding of LDPC codes and has been shown to offer good performance [7]. Before a full description of the belief propagation algorithm is given it is necessary to understand how to convert the received symbols into conditional probabilities.

2.2.1 Calculating Reliabilities on an Additive White Gaussian Noise Channel

The belief-propagation algorithm uses conditional probabilities within a Bayesian network to decode codewords. Subsequently the symbols of the codewords need to be converted into conditional probabilities. One approach to this problem has been presented in [7] and will be recalled here. Assuming a Binary Phase Shift Keying (BPSK) modulator, where the bit 1 is represented by a and the bit 0 is represented by $-a$, the transmitted signal vector t_n is given by

$$t_n = (2c_n - 1)a,$$

where c_n is a vector of binary symbols. For the Additive White Gaussian Noise (AWGN) channel the received vector r_n is the additive combination of the transmitted vector t_n and a Gaussian error vector e

$$r_n = t_n + e.$$

Then the conditional probability of a symbol being the bit 1 given the received symbol is

$$\begin{aligned} P(c_n = 1|r_n) &= \frac{p(r_n|c_n = 1)P(t_n = a)}{p(r_n)} \\ &= \frac{p(r_n|c_n = 1)P(t_n = a)}{p(r_n|t_n = a)P(t_n = a) + p(r_n|t_n = -a)P(t_n = -a)} \\ &= \frac{p(r_n|c_n = 1)}{p(r_n|t_n = a) + p(r_n|t_n = -a)}, \end{aligned} \tag{2.1}$$

given that $P(t_n = a) = P(t_n = -a) = P(c_n = 1) = P(c_n = 0) = \frac{1}{2}$ for a uniformly distributed binary stream. The notations of $P(\cdot)$ and $p(\cdot)$ are used to represent the probability mass (discrete probabilities) and the probability density (continuous probabilities) functions respectively. Equation (2.1) accounts for the aspects of the BPSK modulator assumed. However, the transmission was stipulated across the AWGN channel. Subsequently, the conditional probability on the received symbols, given what was transmitted through the AWGN channel, is given by

$$p(r_n|t_n = a) = \frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{1}{2\sigma^2}(r_n - t_n)^2}$$
$$= \frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{1}{2\sigma^2}(r_n - a)^2}, \tag{2.2}$$

where σ^2 is the noise variance on the channel [7]. Equation (2.2) calculates the conditional probability on the Euclidean distance between the ideal and received symbols t_n and r_n. Combining Equations (2.1) and 2.2 gives

$$P(c_n = 1|r_n) = \frac{1}{1 + \exp^{-2ar_n/\sigma^2}}, \tag{2.3}$$

which can be used to calculate the conditional probability of a bit being 1 given the received symbol. Using Equation (2.3) the conditional probabilities needed in the belief propagation algorithm can be calculated.

2.2.2 Belief-Propagation Algorithm

The belief-propagation algorithm, as mentioned earlier, propagates 'soft-decision' or probabilistic belief information through a tree structure. The tree structure is a bipartite graph which is a graphical representation of the code's parity check matrix, known as a Tanner graph [28]. The parity-check matrix H

$$H_{m,n} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,j} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,j} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ c_{i,1} & c_{i,2} & \cdots & c_{i,j} & \cdots & c_{i,n} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,j} & \cdots & c_{m,n} \end{bmatrix}$$

is used for the construction of the Tanner graph. An example Tanner graph is shown in Figure 2.3.

To perform the decoding, the probabilities are propagated around this tree structure until an exit condition is met or the maximum number of allowable iterations is completed. The belief-propagation algorithm is discussed in many texts, some of which are [3, 7, 29]. The description given for the belief-propagation algorithm here is based on the work from [7].

Before a discussion on the actual algorithm can be presented a few key variables

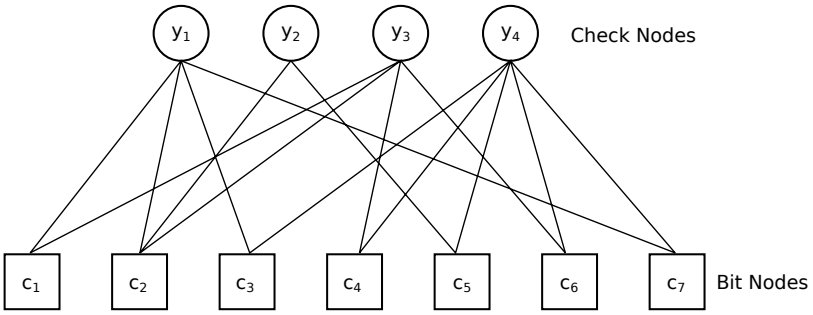

Figure 2.3: An example Tanner graph.

need to be defined. Due to the sparse nature of the H matrix for LDPC codes and their large size it is wasteful to store the entire matrix. Rather, it is far more simple to store a list of non-zero locations within the H matrix. The notation η_m refers to all the non-zero bits within the parity-check matrix for the check at row m. Furthermore, $\eta_{m/n}$ refers to all the non-zero bits in row m that participate in the check except for the nth bit. If the values were inverted, $\eta_{n/m}$, it would refer to all the non-zero checks for the bit in column n except for the mth check. Thus the parity check y_2, from Figure 2.3, can be simplified down to

$$y_2 = \sum_{\eta_n} c_n.$$

Using this notation the belief-propagation algorithm can be fully described; see Algorithm 2.1. After the initialization of all variables involved, the $q_{mn}(0)$ and $q_{mn}(1)$ matrices are populated with the channel *posteriori* information. The algorithm enters a loop, which will iterate until either the codeword is decoded or the maximum number of iterations is completed. During the loop there are three distinct stages; a bit node stage, a check node stage and finally a *pseudo-posteriori* stage. The bit node stage processes the bit nodes from the Tanner graph and calculates the marginal probabilities for those nodes. The marginal probability will then be passed as a message to the check nodes. The check nodes then perform a similar process, marginalising the information from all the bit nodes. The $q_{mn}(0)$ and $q_{mn}(1)$ matrices do not represent a codeword as they are matrices. Thus, to determine a codeword from these matrices the *pseudo-posteriori* stage is performed. The soft-decision codeword generated by the algorithm $q_n(0)$ is then converted to its hard-decision counterpart c'_n, based on the most-likely symbols from the conditional probabilities. If this is a valid codeword the algorithm will exit the loop, if not another iteration is performed.

2.3 Parallel Performance

Parallel algorithms offer performance enhancements over traditional serial algorithms. They separate the processing across multiple independent processors to enhance the

Algorithm 2.1 Belief-Propagation algorithm for the decoding of LDPC codes.

Inputs: Parity-Check Matrix (H),
Channel *Posteriori* Probability Matrix ($p_n(x) = P(c_n = x|r_n)$)
Maximum Number of Iterations (I)
Output: Pseudo-*Posteriori* (p'_n)
Control Variables: exit, Iteration (i)
Initialisation: Set $q_{mn}(x) = p_n(x)\ \forall\ (m,n)$ with $H(m,n) = 1$
while exit $\neq$ *TRUE* **AND** $i \leq I$ **do**
Bit Node Step:
Calculate: $\delta q_{mn} = q_{mn}(0) - q_{mn}(1)$
Calculate:

$$\delta r_{mn} = \prod_{n' \in \eta_{m/n}} \delta q_{mn'}$$

Calculate: $r_{mn}(1) = (1 - \delta r_{mn})/2$ and $r_{mn}(0) = (1 + \delta r_{mn})/2$
Check Node Step:
Calculate:

$$q_{mn}(0) = \alpha_{mn} P_n(0) \prod_{m' \in \eta_{n/m}} r_{m'n}(0) \quad \text{and}$$

$$q_{mn}(1) = \alpha_{mn} P_n(1) \prod_{m' \in \eta_{n/m}} r_{m'n}(1)$$

{where α_n is a normalisation factor chosen so $q_n(0) + q_n(1) = 1$}
Pseudo-*Posteriori* Step:
Calculate:

$$q_n(0) = \alpha_n P_n(0) \prod_{m' \in \eta_n} r_{m'n}(0) \quad \text{and}$$

$$q_n(1) = \alpha_n P_n(1) \prod_{m' \in \eta_n} r_{m'n}(1)$$

{where α_n is a normalisation factor chosen so $q_n(0) + q_n(1) = 1$}
Make a Pseudo-Hard-Decision:
for all $n \in q_n(1)$ **do**
if $q_n(1) \geq 0.5$ **then** Set $c'_n = 1$
else Set $c'_n = 0$
end if
end for
Test Decision:
if $c'_n H^T = 0$ **then** exit = *TRUE*
end if
end while
return $p'_n = q_n(0)$ or $q_n(1)$ {Value chosen is dependant on implementation}

performance. Parallelism is one approach to improve the performance of computing systems as the limitations of the current generation silicon processors are reached [11]. Clearly not all problems can be easily parallelised and often the task of parallelising algorithms can be challenging, if not impossible. In telecommunications, parallelism is being used to improve the performance of typically very costly processes [30–33]. To understand the performance benefits offered by parallel algorithms it is necessary to understand some of the concepts used within the field.

2.3.1 Parallel *Speedup*

The parallel *speedup*, referred to simply as the *speedup*, is the speed increase offered by a parallel algorithm over its fastest serial counterpart [14]. The *speedup* S is given by

$$S = \frac{T_S}{T_P}, \tag{2.4}$$

where T_S is the speed for the fastest serial algorithm and T_P is the speed of the parallel algorithm [14]. If there were no communication costs involved the theoretical maximum *speedup* possible would be equivalent to the number of processors in the parallel processor p. Practically no algorithm can exist without communication to and from a processing unit, so a *speedup* of p is not possible. A more practical approach was presented by Amdahl [34], which takes into account both the number of processors in the parallel system and the percentage of the algorithm that is inherently serial. Using this approach, more accurate estimations on the maximum possible *speedup* can be generated.

2.3.2 Amdahl's Law

Amdahl's Law states that the *speedup* of a parallel algorithm is related to the percentage of the algorithm that is inherently serial [34,35]. It offers a good insight into the performance offered by an algorithm across multiple parallel systems. The possible parallel performance predicted by the algorithm is a relationship between the percentage of the algorithm that is inherently serial α and the number of processors in the parallel system p. Assuming that there are no communication costs involved in the parallel algorithm the processing time for the parallel algorithm T_P is given by

$$\begin{aligned} T_P &= Time\ Processed\ in\ Serial + Time\ Processed\ in\ Parallel \\ &= \alpha T_S + \frac{1-\alpha}{p} T_S \end{aligned} \tag{2.5}$$

Substituting Equation (2.5) into Equation (2.4) yields the *speedup* predicted by Amdahl's Law

$$\begin{aligned} S &= \frac{T_S}{T_P} \\ &= \frac{T_S}{\alpha T_S + \frac{1-\alpha}{p} T_S} \\ &= \frac{1}{\alpha + \frac{1-\alpha}{p}} \\ &= \frac{p}{\alpha(p-1)+1}. \end{aligned} \tag{2.6}$$

Amdahl's Law is a very useful and still relevant analysis tool [35]. It can be used to ascertain the possible parallel performance of an algorithm across multiple parallel systems. One weakness of the algorithm is that during the analysis it is necessary to ignore the communication costs. Thus, on parallel systems where communication costs are of concern the analysis provided by Amdahl's Law is of less use.

2.3.3 Parallel Systems

There are a few types of parallel systems or systems upon which a parallel algorithm can be implemented. The three main types are multi-processor systems, multi-cored systems and cluster systems.

Multi-Processor Systems

An example of a multi-processored system would be a Graphics Processing Unit (GPU). One of the most popular and easily accessible platforms for parallel algorithm development on a GPU system is the Compute Unified Device Architecture (CUDA) developed by Nvidia [36, 37]. The weakness of these GPU systems is the small amount of Random Access Memory (RAM) per core. The small amount of RAM requires information stored in RAM to be intelligently managed. If the RAM is poorly managed, the costs of communication (the transfer of information around the memory) can result in deterioration of the algorithm's performance. Additionally, the architecture of a GPU is such that the cores have to operate in 'lock-step', performing the same operation on each thread.

Multi-Cored Systems

A multi-cored processor is what is found in many of the current generation of desktop computers [38]. These systems do not suffer from the limited RAM or communication costs inherent in the GPU systems, due to the large bus sizes connecting the processors to the RAM. The limitation of the multi-cored processor is the manufacturing process. It is becoming increasingly more challenging to create processors

with more cores due to challenges in the manufacturing process, which limits the number of cores on a single processor [13]. Subsequently, the maximum possible *speedup* is drastically reduced; see Section 2.3. The final type of parallel system is the computing cluster.

Cluster Systems

Computing clusters are a collection of individual computers which communicate between one another to provide a large processing pool. The advantage of a computing cluster is that the cluster can be easily scaled. If the cluster requires additional processing power, more computers can be added to the cluster. Additionally, each of the nodes on a cluster has a large amount of processing power, as it is an entire computer. However, all the communication occurs across a computer network, which is a relatively slow medium for transmitting large amounts of data. As such, algorithms implemented for computing clusters need to perform a large amount of processing on each node of the cluster before communicating that information back to a head server, to limit the use of the network infrastructure. The final issue for a computing cluster is accessibility. A computing cluster is not portable, hence the cluster needs to be remotely accessible. Thus, any machine attempting to leverage the cluster needs a working internet connection, otherwise the cluster is rendered useless.

All of these systems have advantages and disadvantages over one another. The correct selection of the parallel system on which a parallel algorithm is implemented can have a massive impact on the system's performance.

2.4 Chapter Summary

The literature review chapter aimed to prepare the reader for the remainder of the dissertation through a summary of all the relevant works within the necessary fields. The topics of product codes, the belief-propagation algorithm and parallel algorithm performance were presented as they form the basis for the decoder design. All of the concepts discussed in this introductory chapter serve to prepare the reader to tackle the remainder of this work.

3 GRAPH-BASED PARALLEL DECODER

In this chapter one of the two presented algorithms for parallelising the decoding of the product code is discussed. The algorithm presented in this chapter uses a graph structure to identify the optimal codewords within a product code by using the erasure pattern. Optimal is defined here as the minimum number of codewords which, when decoded, correct the maximum number of erasures. This algorithm is a hard-decision erasure decoding algorithm which leads onto the soft-decision algorithm presented in the next chapter. This graph-based decoder was eventually found to contain an NP-complete problem, thus rendering it irrelevant. However, the design presents the fundamental concept of an iterative decoder which uses the independence of the component codewords to parallelise the decoding.

3.1 A Product Code as an x-partite Graph

One of the key features of the graph-based decoding algorithm is the function of translating the erasure patterns in a product code into a graph-based structure. Additionally, the chosen structure needs to be easily leveraged to parallelise the decoding. For this algorithm the chosen structure is a bipartite graph [39]. Consider the example product codeword with erasures shown in Figure 3.1.

	c_1	c_2	c_3	c_4	c_5
r_1	E	0	E	1	0
r_2	1	1	0	E	0
r_3	0	E	E	0	E
r_4	0	1	0	E	1
r_5	0	1	E	1	0

Figure 3.1: An example product codeword with erasures introduced during transmission through the channel.

To convert the product codeword shown in Figure 3.1 into a graph, the rows and columns are assigned identifying values. The values r_j and c_j refer to a row or column codeword respectively within the product codeword. Two sets of nodes are defined; one for the row codewords and the other for the column codewords of the product codeword. These nodes are assigned the r_j and c_j identifiers accordingly. The erasures within the product codeword are then represented by edges within the

graph. A single erasure is common to two codewords and as such a single edge is common to two nodes, one in each set of nodes. It is obvious that an erasure in one row codeword can never be common to another row codeword. Thus, the structure described is clearly a bipartite graph [39–41]. A bipartite graph is a graph with two distinct sets of nodes, where nodes from the one set connect only to nodes within the other set and not to nodes within their own set.

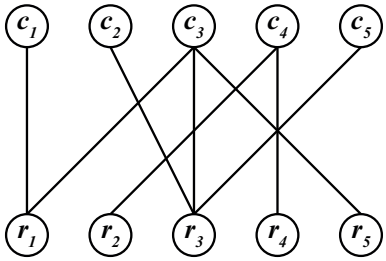

Figure 3.2: The bipartite graph representation of the product code shown in Figure 3.1.

Figure 3.2 shows the graph-based translation of the product codeword shown in Figure 3.1. Using this construction, one can easily see the number of erasures affecting each codeword using each node's degree. The graph construction can also be leveraged to identify the optimal nodes for decoding. For a product code with three dimensions or more, the bipartite graph expands to an x-partite graph, where x is the number of dimensions. Figure 3.3 shows the graph translated from an arbitrary three-dimensional product code.

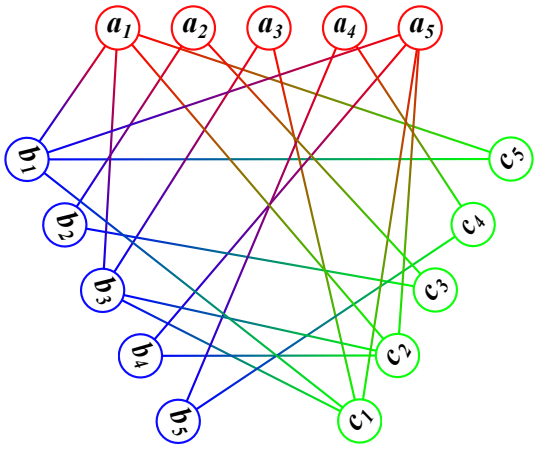

Figure 3.3: The tripartite graph representation of an arbitrary erasure distribution.

3.2 Graph-Based Parallel Decoding Algorithm

The decoding algorithm uses one major feature of the product code to facilitate the parallel decoding. Each of the component codewords in the product codeword has been independently encoded. The encoding process is an i.i.d. process, thus each

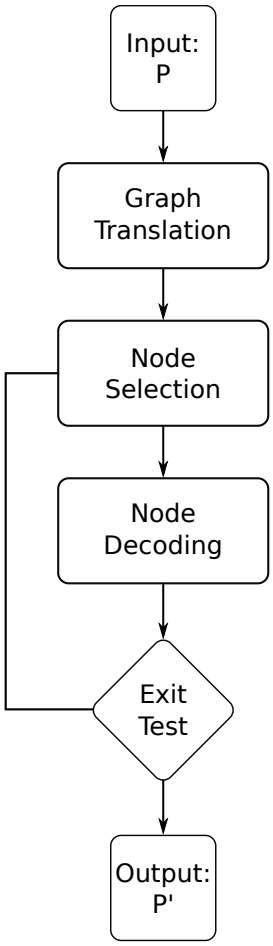

Figure 3.4: An overview of the graph-based decoding algorithm.

decoding can be performed independently of the others [5]. Subsequently, multiple decodings can occur simultaneously, thereby enhancing the decoding time performance. The graph-based decoding algorithm aims to enhance the decoding time performance by efficiently decoding codewords in parallel. During the decoding the optimal nodes[1] are selected from the graph which, when decoded, correct the maximum number of erasures with the minimum number of codeword decodings. So not only is the algorithm capable of processing in parallel, but it also does this in the most efficient manner.

The algorithm has three main components; the node selection, the node decoding and the exit test. Figure 3.4 shows the interaction of all these components. A product codeword P is received from the channel and begins the decoding process. The decoding algorithm is an erasure decoding algorithm, thus simulations would need to be performed on the Binary Erasure Channel (BEC) [4]. P is then translated into a graph through the process outlined in Section 3.1. The optimally decodable

[1] A node in the graph refers to a codeword in the product codeword

nodes[2] are then selected. The selected nodes then undergo the relevant decoding algorithm based on the encoding technique used. After the decoding, the graph is analysed to see whether any more erasures still exist in the product codeword. If so the algorithm reiterates through the node selection and node decoding stages until all the erasures are corrected. If not, the algorithm simply exits. All of the three critical aspects of the algorithm will now be discussed.

3.2.1 Node Selection

The node selection stage selects the nodes for the node decoding stage. An initial naive selection is made. This naive selection includes all nodes in the graph that have a degree lower than the minimum Hamming distance d_{min} of the chosen component code. Equation (3.1) shows the relationship between d_{min} and the number of errors t or erasures e correctable by the code [7,29].

$$d_{min} \geq 2t + e + 1 \tag{3.1}$$

For an erasure correcting code t goes to zero. Hence, the codes used will be capable of correcting $d_{min} - 1$ erasures. Assuming a d_{min} of three then, using the graph shown in Figure 3.2, all the nodes with a degree less than or equal to two are selected, as illustrated in Figure 3.5.

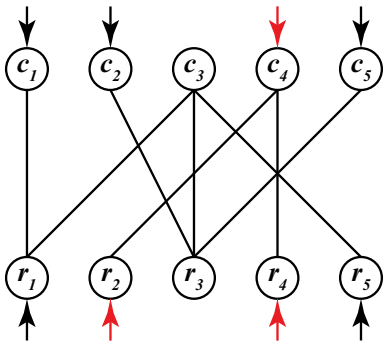

Figure 3.5: Initial naive node selection for the first iteration.

If all the currently selected nodes were to be processed, many irrelevant decodings would be performed, because all the erasures are common to both a row and column codeword. Subsequently, each erasure can be corrected by decoding either the row or column codewords. Thus, only the nodes which provide an optimal decoding are selected. It is at this point that the algorithm encounters its fatal flaw. Consider the nodes r_2, r_4 and c_4. All three nodes are decodable, as their degrees are below the d_{min} threshold. However, the two edges linking these three nodes can be removed by decoding any two of the nodes in the set or by decoding only c_4. Hence the

[2] A node in the graph refers to a codeword in the product codeword

decoding of c_4 is the optimal decoding for that set. The only way to determine this is to traverse the entire set and check the degree and connections of every node in the set and then select the optimal nodes. In the example given this is a trivial task, but on large data sets which are processed by a computer the problem is no longer trivial and can become very costly. The entire graph may need to be traversed to determine the optimal decoding for only a small sub-set of nodes. This problem will be discussed further in Section 3.3. Figure 3.6 illustrates the optimal node selection for the first decoding iteration. The depicted optimal selection may not be the only optimal selection for the given graph. Optimal for this algorithm is defined simply as the minimum number of decodings being performed to correct the maximum number of errors. As such, multiple optimal selections may exist for one graph.

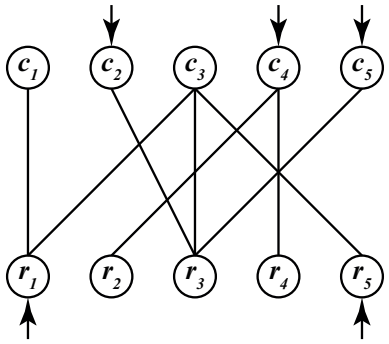

Figure 3.6: The optimal node selection for the first iteration.

3.2.2 Node Decoding

The node decoding stage performs the decoding of all the nodes selected in the node selection stage. The interesting feature of this algorithm is that at this stage any relevant decoding algorithm can be used. The graph-based decoding algorithm places no limitations on the technique chosen for the component codes. However, as this is an erasure decoding algorithm, the technique chosen should be applicable to erasure correction. Thus, any coding techniques, and subsequent decoding techniques, that best meet the requirements of the intended application can be selected.

After all the selected nodes have been processed the graph needs to be updated. Thus, all the edges that previously corresponded to erasures need to be pruned. The resulting graph, shown in Figure 3.7, has had all the relevant edges pruned. The graph must then be checked to see if the decoding has completed.

3.2.3 Exit Test

The exit test is the simplest of the three component phases. The graph is checked to see if it contains any nodes with a non-zero degree. If there are any nodes which have a non-zero degree the algorithm enters an additional decoding iteration. When

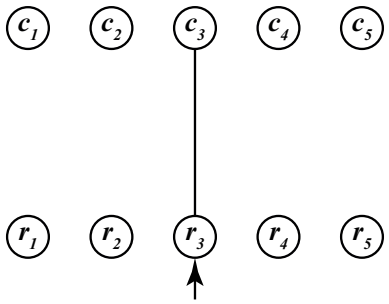

Figure 3.7: Nodes remaining after the node decoding stage for the first iteration.

all the edges have been removed (when all the erasures have been corrected), the algorithm exits. The need for multiple iterations defines this decoding algorithm as an iterative hard-decision erasure decoding algorithm.

3.3 Algorithm Flaws

As previously mentioned, the algorithm's flaw lies in the selection of the optimal nodes for decoding. Although for small problem sets the optimal node selection can easily be selected by a human viewing the nodes, for large problem sets a human can no longer make the optimal selection. Thus, an algorithm would need to be developed to solve this problem. The task of identifying the optimal nodes can be achieved through tree traversal. However, traversing the tree is an inherently serial process. This task could severely decrease the *speedup*, as predicted by Amdahl's Law [34], as it becomes more cumbersome for larger problem sets. Consequently, this algorithm was never taken to prototype, as it was deemed unlikely to meet the research aims considering so large a flaw was found. Upon consideration, if the algorithm were to be implemented on larger, more complex parallel systems, such a large serial component would render the advantages offered by the many processors useless. This optimisation stage, which makes the algorithm more efficient, can thus result in a loss in performance. This is not to say that the graph-based algorithm could not be implemented and offer performance improvements, but rather that the improvements might be somewhat limited. Additionally, if the optimisation stage were to be ignored, this algorithm would still offer decoding time performance improvements over other serial algorithms. However, due to the limited novelty of this solution a more advanced soft-decision algorithm was designed that takes into account the limitations of specific parallel architectures. Hence, for the next design, rather than deliberating over wasted processing time, focus is placed on limiting the serial processing time. The key feature of the product code to take into the next decoding is that each of the component codewords in the product codeword is independent and can be decoded independently. Also, because information can be shared between the separate component codeword decodings an iterative approach would offer the best performance as the shared information needs a few iterations to

propagate throughout the product code.

3.4 Chapter Summary

In this chapter, the first technique for the parallel decoding of product codes is proposed. This algorithm attempts to efficiently decode a product code by focusing on the patterns evident in the product codeword's erasures. This is done by mapping the erasures within the product codeword to a bipartite graph structure. The graph structure can then be used to easily select the optimal nodes for decoding. In this context, optimal refers to the decoding of the minimum number of codewords which corrects the maximum number of erasures in the product codeword. The algorithm presented contains four distinct parts. Initially an erasure pattern needs to be mapped to a graph structure. Then from that graph, the optimal codewords for decoding are selected. The codewords are then decoded, using the relevant decoding technique. Finally, the graph is tested to see if any more erasures are present; if so the graph-based algorithm reiterates through the process or if not it exits. This algorithm could technically have been used with product codes using any component coding techniques, including LDPC codes. However, the algorithm never went further than the design phase, as it was found that to select the optimal nodes is a non-trivial problem that is an NP-complete problem.

The main lessons learned in the design of the graph-based algorithm are that for parallel systems it is often not necessary to go through optimisation processes and that the independence of the component codewords offers the best avenue for parallelisation. The optimisation processes can typically only be processed in serial and can severely limit the performance benefits of parallelising the algorithm. Subsequently, it is better to have an algorithm which maximises the parallel aspects of the algorithm to offer better performance. An important feature of this algorithm to take forward is the ability of the graph-based algorithm to make use of the independence of the component codewords to facilitate the parallel decoding. This feature of the product code is critical to the development of further parallel product code decoders. Additionally, the algorithm makes no provision for specific parallel architectures, and as such would be challenging to implement on all parallel architectures. Finally, due to the inherent sharing of information between the component codewords within a product codeword, the best decoding approach is an iterative decoder to allow the shared information to propagate throughout the codeword.

4 PARALLEL BELIEF-PROPAGATION DECODER

During the design of the graph-based decoder presented in Chapter 3, it was found that attempting to implement an efficient decoder can severely reduce the performance benefits offered by a parallel system. A new decoder is presented, which aims to maximise the number of simultaneous processes and minimise the serial processing time, which in turn will maximise the parallel *speedup* [34]. As opposed to the iterative hard-decision erasure decoder presented in Chapter 3, this decoder is an iterative soft-decision error correcting decoder. This decoder builds on the belief-propagation algorithm presented by Judea Pearl [10]. Some of the ideas used for the development of this decoder come from the work into the decoding of the product code by both Reddy and Pyndiah [17–19, 23]. The belief-propagation algorithm is the only option for the decoding of LDPC product codes, as both Chase- and trellis-based decoders have impractical complexities for LDPC codes; see Section 2.1.3. Chase-based decoders can, in practice, only decode codes with small minimum Hamming distances due to the size of the sub-set having to be exhaustively searched [22]. However, LDPC codes have very high minimum Hamming distances and as a result Chase decoders are impractical for the decoding of LDPC codes. Trellis-based decodes suffer from increased complexity for long length codes. However, LDPC codes have very long lengths, rendering trellis-based decoding impractical as well. Consequently, the only option is to use a belief-propagation based decoder, as it provides a complexity that increases linearly with length.

As discussed in Section 2.2, the belief-propagation algorithm is a statistical inference algorithm for determining the marginal probabilities across a Bayesian network. In channel coding the belief-propagation algorithm can be used to decode any coding technique that can be represented by a Tanner graph. The only proviso on this is that the Tanner graph contains no cycles with a girth of 4. This fact limits the usability of the belief-propagation decoding algorithm as a decoding algorithm for most coding techniques. It is unlikely for short codes such as Hamming codes and Reed-Solomon codes to have a Tanner graph not of girth 4. This is due to the high density of ones within the parity-check matrix of these codes. Due to the sparsity of ones in the LDPC code's parity-check matrices, these codes have been shown to be successfully decoded with the belief-propagation decoding algorithm [7]. Subsequently, the belief-propagation decoding has been shown to offer very good performance for the decoding of LDPC codes [42–44]. A product codeword constructed out of LDPC codes would be very large containing many component codewords. Decod-

ing an LDPC product code in a typical serial fashion is very complex, and makes the LDPC product code impractical in real world implementations. By parallelising the decoding of these codes, the decoding performance can be substantially improved, making these very powerful codes a viable solution for real world implementations.

As previously mentioned in Chapter 3, the key feature of a product codeword is the independence of its component codewords [5,7,17]. The decodings can be performed independently, because each of the encodings is performed by an i.i.d. encoding process. This fact combined with the high number of independent component codes within an LDPC product code results in a large number of decodings that can occur simultaneously. Thus, the product code is ideal for parallelisation as the *speedup* achievable is related to the number of processors across which the problem can be split [14]. This chapter presents the design for a two-dimensional belief-propagation decoder for the decoding of LDPC product codes. Although, all the discussions in this chapter focus on a two-dimensional product code, the work and concepts can be easily expanded to encompass more dimensions.

4.1 Overview

The parallel belief-propagation decoder parallelises the belief-propagation decoding process for the product code by adding in additional stages to the traditional belief-propagation algorithm. The codeword reliability estimation, belief-aggregation and exit testing stages are added to facilitate the parallel decoding of the product code. The overall decoder follows the structure outlined in Figure 4.1. The decoder takes an initial product codeword P which has been transmitted through some channel. To maximise the number of component codewords processed in parallel, the product codeword P is duplicated. The extra product codeword is transposed P^T to allow a single belief-propagation process to perform the decodings for both the row and column codewords within the product codeword.

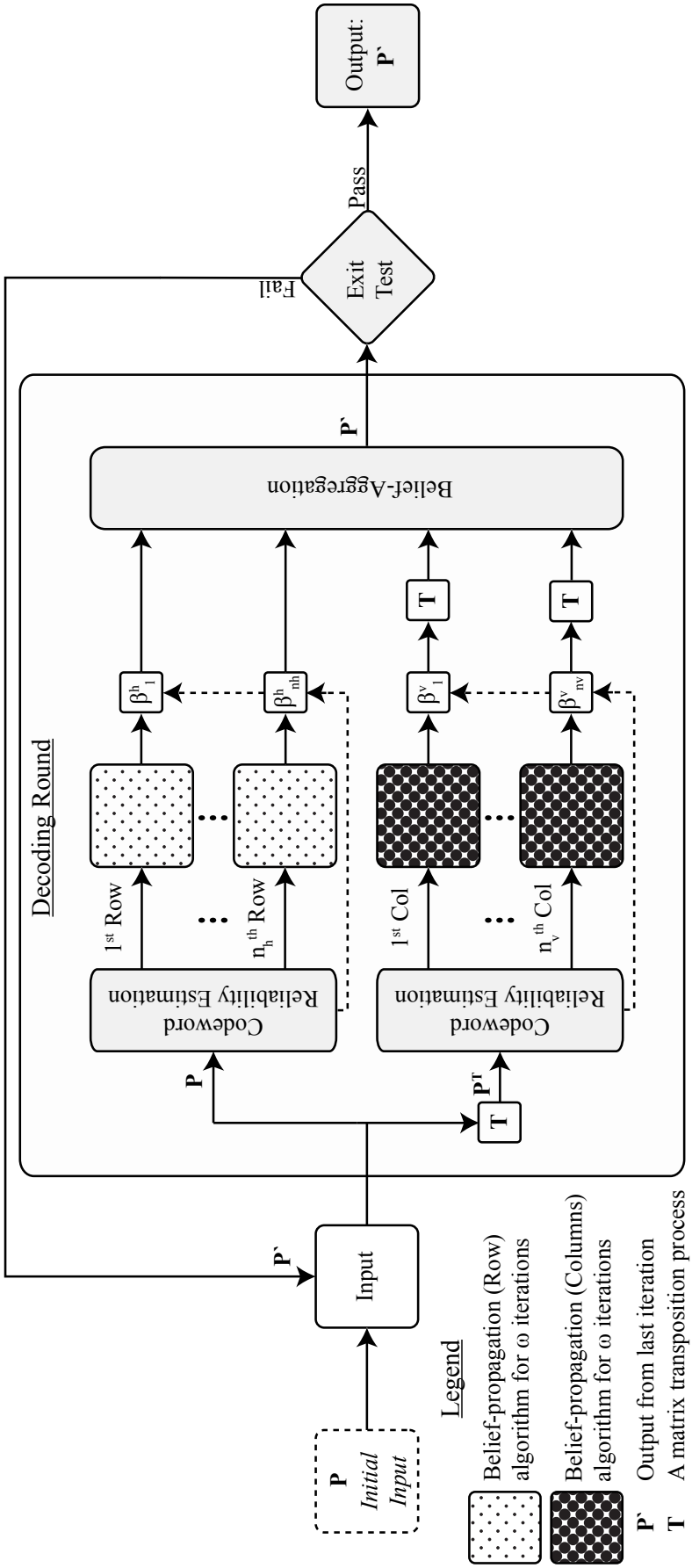

Figure 4.1: A flow diagram describing the entire parallel belief-propagation decoder.

Before the decoding can actually begin it is necessary to estimate a codeword's reliability. The codeword's reliabilities need to be estimated, because there are two product codewords, P and P^T, which need to be recombined post-decoding. If the information is considered equally, the recombination can result in the unreliable information weakening the reliability of the entire product codeword. A reliability vector is generated for each codeword within the product codeword. The two product codewords then undergo the belief-propagation decoding algorithm simultaneously. The belief-propagation decoding process for each codeword undergoes ω belief-propagation iterations, called inner loops here, before stopping. After the defined number of inner loops the product codewords need to be recombined. The codeword's conditional probabilities are then scaled by the reliability vector, determined in the codeword estimation stage, before being recombined.

The belief-aggregation stage recombines the beliefs from the two product codewords. The use of the weighting vectors in the belief-aggregation process mitigates the influence of the unreliable component codewords on the resulting product codeword. The new product codeword P' undergoes an exit test to evaluate whether the decoding process has been successful and a valid codeword has been found. If the exit test fails, the product codeword re-enters the entire process - an outer loop Ω - once again. If P' passes the exit test, the new product codeword P' is outputted from the decoder. Each of the stages with some of the conceived permutations will be discussed further in this chapter. The set of presented permutations of each stage by no means encompasses all the conceivable ways in which to perform that stage.

The product code is a two dimensional binary product code. The product code is structured as follows;

$$P = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,j} & \cdots & c_{1,n_h} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,j} & \cdots & c_{2,n_h} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ c_{i,1} & c_{i,2} & \cdots & c_{i,j} & \cdots & c_{i,n_h} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ c_{n_v,1} & c_{n_v,2} & \cdots & c_{n_v,j} & \cdots & c_{n_v,n_h} \end{bmatrix},$$

where $c_{i,j}$ is a symbol in the code, n_v and n_h are the lengths of the row and column codewords and i and j are the row and column positions. For the remainder of this chapter all discussions will be made with reference to this product code structure.

4.2 Codeword Reliability Estimation

To maximise the number of simultaneous belief-propagation decoding processes, the product codeword P is duplicated. One of the product codewords is used for the row and the other for the column codewords' decodings. After the decoding, these two product codewords need to be recombined to create a single output product codeword. At each position in the product codeword there are now two information sources, one from each of the product codewords. These sources must be recombined to create the decoded product codeword. During the recombination process, if both values are considered equally reliable, an unreliable value can severely weaken their combined reliability. Subsequently, the codeword reliability estimation stage estimates the reliability of all the codewords before recombination. Then during recombination, the unreliable information sources can be ignored or scaled to reduce their impact on the final product codeword. This forms the basis for codeword reliability estimation.

Codeword reliability estimation is useless without some system to actually remove the unreliable information. This is the function performed by the belief-aggregation stage; discussed in Section 4.4. The combination of these two processes is responsible for the removal of this unreliable information from the final product codeword. Neither of these two processes can be performed independently.

4.2.1 Weighting Techniques

Two different techniques for codeword reliability estimation are presented; reliability estimate weighting and no weighting. Both of these techniques will generate a weighting factor ϕ, which will be used to create a weighting vector for a product codeword.

Reliability Estimate Weighting

The first technique constructs a weighting factor from the conditional probabilities of the received signal vector r_n; see Section 2.2. The reliability factors are used in the belief-propagation algorithm to decode the codeword. These values can easily be leveraged to provide an estimate of the reliability of a given codeword. Consider the received constellation points after being transmitted across a channel by a BPSK modulator, shown in Figure 4.2. Unreliable constellation points are those which lie at a point equidistant to both ideal constellation points, in this case 1 and -1. Using the logic described in Section 2.2, these constellations would have a reliability of approximately 0.5, as they are equally likely to be either of the ideal symbols.
A codeword consisting largely of these unreliable constellation points would subsequently provide a very low confidence of successful decoding, as the decoder would

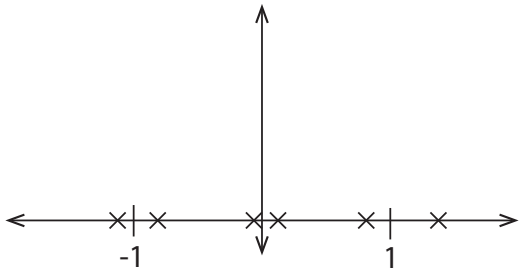

Figure 4.2: A set of arbitrary received symbols on a constellation diagram.

struggle to converge to the correct codeword. Consequently, the weighting vector ϕ for this technique is given as

$$\phi_i = \frac{2}{n_h} \sum_{j=1}^{n_h} max(c_{i,j}^0, c_{i,j}^1) + 1, \tag{4.1}$$

where Equation (4.1) gives the weighting factors for the horizontal dimension. For the vertical dimension

$$\phi_j = \frac{2}{n_v} \sum_{i=1}^{n_v} max(c_{j,i}^0, c_{j,i}^1) + 1 \tag{4.2}$$

is used. Equations (4.1) and (4.2), calculate the average of the maximum conditional probabilities for each bit in a codeword. To ensure minimal impact on the final product codeword, the unreliable codewords need to have weightings close to zero. However, using just a plain average of the highest reliability factor for each bit would generate a weighting factor of 0.5 for unreliable codewords. Consequently, the weighting factors are normalised to 0.5 to ensure that an unreliable codeword generates a weighting factor close to zero.

No Weighting

As the name suggests, this technique assigns no weightings. Rather it assumes that each of the codewords is viewed as equally reliable by assigning a unity factor to all the codewords.

4.2.2 Construction of a Weighting Matrix

Each bit within a codeword is a combination of two weighting factors. Thus, the two weighting vectors created

$$\phi^h = \begin{bmatrix} \phi_1^h \\ \phi_2^h \\ \vdots \\ \phi_i^h \\ \vdots \\ \phi_{n_h}^h \end{bmatrix}, \tag{4.3}$$

and

$$\phi^v = \begin{bmatrix} \phi_1^v & \phi_2^v & \cdots & \phi_i^v & \cdots & \phi_{n_v}^v \end{bmatrix} \tag{4.4}$$

for the horizontal and vertical dimensions need to be combined into a weighting matrix to reflect each bit within the product codeword. To ensure that the combination of these probabilities maintains the integrity of the probabilities, a normalisation factor is introduced. This factor normalises the weighting factors to one which guarantees that the conditional probabilities will still sum to one. The normalisation factor is defined as

$$\rho_{i,j} = \frac{1}{\phi_i^h + \phi_j^v}$$

and is constructed to normalise the weighting factor matrices, where ϕ_i^h and ϕ_j^v are the horizontal and vertical weighting factor vectors. The resulting matrix

$$\rho_{i,j} = \begin{bmatrix} \frac{1}{\phi_1^h+\phi_1^v} & \frac{1}{\phi_1^h+\phi_2^v} & \cdots & \frac{1}{\phi_1^h+\phi_j^v} & \cdots & \frac{1}{\phi_1^h+\phi_{n_v}^v} \\ \frac{1}{\phi_2^h+\phi_1^v} & \frac{1}{\phi_2^h+\phi_2^v} & \cdots & \frac{1}{\phi_2^h+\phi_j^v} & \cdots & \frac{1}{\phi_2^h+\phi_{n_v}^v} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ \frac{1}{\phi_i^h+\phi_1^v} & \frac{1}{\phi_i^h+\phi_2^v} & \cdots & \frac{1}{\phi_i^h+\phi_j^v} & \cdots & \frac{1}{\phi_i^h+\phi_{n_v}^v} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ \frac{1}{\phi_{n_h}^h+\phi_1^v} & \frac{1}{\phi_{n_h}^h+\phi_2^v} & \cdots & \frac{1}{\phi_{n_h}^h+\phi_j^v} & \cdots & \frac{1}{\phi_{n_h}^h+\phi_{n_v}^v} \end{bmatrix},$$

can then be combined with the weighting vectors to create the weighting matrices

$$\beta^h = \begin{bmatrix} \rho_{1,1}\phi_1^h & \rho_{1,2}\phi_1^h & \cdots & \rho_{1,n_v}\phi_1^h \\ \rho_{2,1}\phi_2^h & \rho_{2,2}\phi_2^h & \cdots & \rho_{2,n_v}\phi_2^h \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{n_h,1}\phi_{n_h}^h & \rho_{n_h,2}\phi_{n_h}^h & \cdots & \rho_{n_h,n_v}\phi_{n_h}^h \end{bmatrix}, \tag{4.5}$$

and

$$\beta^v = \begin{bmatrix} \rho_{1,1}\phi_1^v & \rho_{2,1}\phi_1^v & \cdots & \rho_{n_h,1}\phi_1^v \\ \rho_{1,2}\phi_2^v & \rho_{2,2}\phi_2^v & \cdots & \rho_{n_h,2}\phi_2^v \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1,n_v}\phi_{n_v}^v & \rho_{2,n_v}\phi_{n_v}^v & \cdots & \rho_{n_h,n_v}\phi_{n_v}^v \end{bmatrix}. \tag{4.6}$$

These matrices are used to scale the product codewords after the belief-propagation decoding process. Each of the rows in the matrices is dot multiplied against a codeword. Therefore, given an arbitrary codeword, the codeword would be scaled by its corresponding weighting vector (a row from the weighting matrix),

$$c_n \cdot \begin{bmatrix} \rho_{1,i}\phi_i^v & \rho_{2,i}\phi_i^v & \cdots & \rho_{n_h,i}\phi_i^v \end{bmatrix}.$$

The scaled product codewords are then passed to the belief-aggregation stage to combine the product codewords from the row and column decoding processes.

4.3 Belief-Propagation

The belief-propagation stage forms the basis for the entire decoder. The belief-propagation algorithm, as discussed in Section 2.2, is extensively discussed in literature and as such it will not be expanded upon further; see for example [3, 7, 10, 29]. However, the parallel belief-propagation stage does make some minor changes to the operation of the generalised belief-propagation algorithm. The changes introduced modify the existing algorithm slightly and as such it is expected that the reader is familiar with the original algorithm; see Section 2.2.

One of the interesting features of a product code, as well as being the feature that is leveraged to parallelise the belief-propagation algorithm, is the independence of the component codewords. Because each codeword is encoded by an i.i.d. process, the decoding processes for each codeword can be performed independently. Subsequently, all the codewords within a product code can be decoded simultaneously on a parallel processor. This feature of a product code forms the basis for the parallel decoder presented here. However, for a two dimensional product code, each bit within a product code is linked to two separate encoding processes. Typically de-

coding would proceed first with the horizontal and then with the vertical decoding in a serial manner. If one assumes that all the codewords in a single dimension can be processed in parallel, the maximum theoretical *speedup* S would be limited to

$$S \leq n$$

if $n_h = n_v = n$. The *speedup* is limited to n, because only the n horizontal codewords are processable in parallel, as the vertical decodings depend on the horizontal decodings. To increase the maximum theoretical S, the product codeword P is duplicated before processing. The duplicate is then transposed (denoted by P^T). The transposition process exposes the vertical codewords to an identical belief-propagation process, as is used on the horizontal codewords. The duplication and transposition then increase the maximum theoretical S to

$$S \leq 2n,$$

assuming that $n_h = n_v = n$. The *speedup* is double the serially processed product code. This duplication process is introduced in the belief-propagation stage to maximise the parallelism of the decoder. All the additional processes exist to facilitate the recombination of the P and P^T matrices after undergoing the belief-propagation algorithm. By making these changes the maximal theoretic *speedup* approaches what would appear to be the maximum possible *speedup* for the decoding of a product code.

The typical belief-propagation algorithm consists of four stages that are iterated until either a successful decoding is achieved or the maximum number of iterations is reached; see Section 2.2 for more. Clearly if this approach were used within the parallel decoder, there would be some codewords that would take more iterations than others. More problematically, there would be some codewords which are undecodable and would run to the maximum number of iterations. The parallel decoder cannot move to the next decoding stage until all the decodings are complete. Therefore the entire decoding process would be stalled by a single slow decoding. In the case of the product code this is particularly short-sighted, as the decoding of an undecodable codeword can be assisted by the other codewords with which it shares bits. As such, the belief propagation algorithm has been modified to remove this eventuality. The modifications focus around the iteration control. Firstly, the exit test which tests for successful decoding is removed and placed later in the decoding process, which will be discussed in Section 4.5. As the algorithm now has no means of determining when to exit, a defined number of belief-propagation iterations ω is defined for the decoding process. This means that each time a codeword undergoes

the belief propagation stage, it undergoes ω iterations before the *pseudo-posteriori* is calculated. The algorithm modifications are shown in Algorithm 4.1. An additional advantage of a defined ω lies in the nature of some of the parallel processors where all the processors need to perform an identical process in 'lock-step', such as a GPU. As such, each codeword decoding could not occur as a unique instance with its own unique parameters. To ensure that all the codeword decodings remain in 'lock-step', an identical number of belief-propagation iterations is performed per outer loop. This modification allows the decoder to be more easily implemented across multiple different parallel frameworks.

Algorithm 4.1 A modified belief-propagation algorithm for the parallel decoding of LDPC product codes.

Inputs: Parity-Check Matrix (H),
Channel *Posteriori* Probability Matrix ($p_n(x) = P(c_n = x|r_n)$)
Number of Iterations (ω)

Output: Pseudo-Posteriori (p'_n)
Control Variables: Iteration (i)
Initialisation: Set $q_{mn}(x) = p_n(x)\ \forall\ (m,n)$ with $H(m,n) = 1$
while $i \leq \omega$ **do**
Bit Node Step:
Calculate: $\delta q_{mn} = q_{mn}(0) - q_{mn}(1)$
Calculate:

$$\delta r_{mn} = \prod_{n' \in \eta_{m/n}} \delta q_{mn'}$$

Calculate: $r_{mn}(1) = (1 - \delta r_{mn})/2$ and $r_{mn}(0) = (1 + \delta r_{mn})/2$
Check Node Step:
Calculate:

$$q_{mn}(0) = \alpha_{mn} P_n(0) \prod_{m' \in \eta_{n/m}} r_{m'n}(0) \quad \text{and}$$

$$q_{mn}(1) = \alpha_{mn} P_n(1) \prod_{m' \in \eta_{n/m}} r_{m'n}(1)$$

▷ where α_{mn} is a normalisation factor chosen so $q_{mn}(0) + q_{mn}(1) = 1$
end while
Pseudo-Posteriori Step:
Calculate:

$$q_n(0) = \alpha_n P_n(0) \prod_{m' \in \eta_n} r_{m'n}(0) \quad \text{and}$$

$$q_n(1) = \alpha_n P_n(1) \prod_{m' \in \eta_n} r_{m'n}(1)$$

{where α_n is a normalisation factor chosen so $q_n(0) + q_n(1) = 1$}
return $p'_n = q_n(0)$ or $q_n(1)$ {Value chosen is dependent on implementation}

This leads on to the fact that the performance of the parallel decoder will be directly linked to the number of iterations chosen. Choosing too low an ω can result in the belief-propagation having a lack of sufficient iterations to infer a good marginal distribution across the Bayesian network. As a result, the beliefs have not had sufficient time to propagate throughout the Bayesian network and a poor *posteriori* probability is generated. The poor *posteriori* probability will then be used as a *priori* for the next outer loop, negatively impacting the efficacy of that decoding, and so on. If an overly high ω is chosen, excessive decoding rounds can be performed. Although this does not affect the decoding BER performance, it can result in higher processing costs in achieving a successful decoding. Thus it is evident that the performance of the parallel decoder is inherently linked to the number of belief-propagation iterations ω performed in a single outer loop.

4.4 Belief-Aggregation

The belief-aggregation stage recombines the information from the row and column, P and P^T, product codeword decodings into a single unified product codeword. This stage generally operates in conjunction with the codeword reliability estimation stage. The belief-aggregations stage uses the weighting information to limit the impact of unreliable information on the final product codeword P'. The final product codeword generated after the belief-aggregation stage P' is then tested for successful decoding. If the product codeword P' has been successfully decoded, P' is what is outputted from the parallel belief-propagation algorithm. The belief-aggregation technique designed, uses an average to recombine the two separate product codes.

The averaging technique simply performs an average on the two product codewords (P and P^T). The averaging technique assumes that all the information in the product codewords is valuable. Rather than the unreliable codewords being ignored, they are still incorporated into the final product codeword. The type and behaviour of the averaging techniques depend heavily on the codeword reliability estimation stage. If the codeword reliability estimation stage is chosen to provide no weightings, when the product codewords are averaged it would result in a normal average. If a weighting technique that weights the product codewords before aggregation is chosen, the average will then be a weighted average. Both techniques follow the general form for the averaging process

$$P' = \beta_i^h \cdot P_i + (\beta_j^v \cdot P_j^T)^T.$$

The weighting factors used in this stage determine whether the average is unweighted or weighted. Due to the normalisation stage introduced in the generation of the β matrices, a simple summation acts as an averaging process.

Unweighted Average A unweighted average is generated when the codeword reliability estimation is set to generate no weightings. Thus the β matrices are reduced to unity. A unweighted average ignores unreliable codewords with the assumption that sufficient reliable information still remains within the product codeword and the decoding will still converge.

Weighted Average A weighted average is generated when any of the codeword reliability estimation techniques other than no weighting are used. A weighting technique assumes that by reducing the amount of unreliable information within the Bayesian network, a decoding can be more successful. Not only would the decoding converge more rapidly towards the correct codeword, but would also require less iterations for a successful decode, as there is a higher prevalence of reliable information.

4.5 Exit Test

The exit test stage is responsible for determining when the decoding process is complete and exiting the decoding. This stage is included due to the exit testing having been removed from the base belief-propagation algorithm to facilitate more predictable parallel decoding; see Section 4.3. Typically, an exit test in the belief-propagation algorithm would be performed by calculating the syndrome for all the codewords within the product codeword. If the syndromes for all the component codewords have

$$\sum_{j=1}^{n} c_j H^T = \bar{0}, \tag{4.7}$$

then the algorithm exits. The shortcoming of this technique is the requirement that $n \times m$ multiplications be performed to determine the syndrome for a single component codeword. For the long length of LDPC codes combined with the number of component codes within the product code, this can quickly become a very costly process. One advantage of the technique is that the syndrome calculations on the product code can be processed in parallel, in much the same way as the component code decodings. However, the calculation for a single codeword still needs to be performed serially, which can be slow for long LDPC codes. Furthermore, the requirement that the exit test be performed for every outer loop can be very limiting to the product code's overall decoding time performance, especially for decodings that could have a high Ω. Amdahl's Law suggests that by increasing this serial processing time, the maximum *speedup* of the decoder becomes more limited [14, 34]. Additionally, if all the codewords are to be decoded by a single belief-propagation algorithm on a parallel processor such as a GPU, the processing would need to be performed in lock-step. Thus, individual decodings would not be able to exit until

all the decodings had completed. As such it would be better to remove the exit test from each individual codeword's decoding and to perform an exit test on the entire product code. This argument supports the decision made in Section 4.3.

The exit testing process does not directly affect the decoding BER performance of the product code. However, a poorly defined exit test can result in early or late exits. If the decoder exits early from the decoding (before the decoding can complete) not all errors will be corrected, which will increase the decoding BER of the product code. Conversely, if an exit test detects late in the decoding process that a product code is undecodable, processing time is wasted slowing down the decoding. To alleviate these issues some alternative techniques have been suggested for the exit test process, which aim to reduce the processing cost of the exit test and facilitate the early detection of undecodable codewords.

4.5.1 Syndrome

As mentioned, the syndrome technique is the standard test for successful decoding for a linear block code. However, it can be quite costly for large codes. The advantage of using a syndrome technique is that it provides a guaranteed method for ascertaining whether a product codeword has been successfully decoded. This makes the syndrome technique a better option when using a technique that needs a more defined exit control. The limitation of this technique is that it provides no means of early detection of undecodable product codewords. Hence the decoder will continue decoding an undecodable codeword, wasting significant processing time.

4.5.2 Defined Ω

This is a very naive technique. Rather than using a process to determine whether or not the decoding has been completed, a defined number of outer loops Ω is specified for all decodings. It is immediately obvious that this approach is short-sighted. The number of outer loops required will be directly related to the SNR on the channel and for harsher or more mild SNR a higher or lower Ω would be required. Due to this technique's obvious limitations it was never taken to prototype, as it provides no justifiable benefit over the standard syndrome test.

4.5.3 Variance Thresholding

The next technique uses a different approach altogether. Rather than looking at the individual codewords within the product codeword to see if all the component codewords have been decoded, variance thresholding looks at metrics on the product codeword itself. This technique makes the assumption that when the product codeword has been successfully decoded, the decoding will have converged to a stable state. Thus, metrics taken on the product codeword should remain stationary across

two or more outer loops. It is immediately evident that this technique will require at least a single additional outer loop before an exit can be made. However, this technique can also provide a means for early exits, as deviation from convergent behaviour could be indicative of a failed decoding. This technique is less costly than the syndrome technique presented, as the entire product codeword is considered instead of a single codeword within the set. Therefore less processing needs to be performed.

Variance thresholding uses the variance of the product codeword's conditional probabilities to determine when to exit. Consider a stream of binary symbols t_n, which is transmitted across an AWGN channel using a BPSK modulator. The conditional probabilities on the received symbol stream r_n can be determined using the process outlined in Section 2.2. The logic dictates that for successive outer loops the probabilities, conditional to 0 or 1, should drift towards the edges (probabilities of either 0 or 1). Thus, the variance on the product codeword should at some point reach a point of stability, either when the product codeword is fully decoded or when the decoding has stabilised to some point. Lemma 4.5.1 shows that this can be proven to be true.

Lemma 4.5.1 (Variance convergence)**.** *The variance for an arbitrary series of conditional probabilities of an* i.i.d. input symbol converge to 0.25 for a successful decoding.

Proof. For a successful decoding $P(c_n = 1|r_n)$ tends towards either zero or one. It is known that r_n is conditional to a uniformly distributed i.i.d. binary data stream t_n. The mean μ of that binary stream t_n would be 0.5. Subsequently, following a successful decoding r_n can be considered identical to an i.i.d. binary stream which has a variance of 0.25.

□

Under real testing Lemma 4.5.1 can be shown to be valid with the results for some successive decoding rounds showing convergence towards 0.25, as seen in Figure 4.3. For codewords that are decodable the variance stabilises to 0.25. For decodings that begin to exceed a code's error correcting capability, the variance fails to reach 0.25 and fluctuates continuously.

Although the variance converging towards 0.25 provides a very good indication of when a decoding is complete, it does not provide any means for early detection of unstable or failed decodings. If one assumes that a non-convergent behaviour is indicative of a failed decoding, a few approaches can be used to assist in the early detection of a failed decoding. An initial approach would be to consider the degree to which the system is chaotic by testing for positive Lyapunov exponents [45]. The limitation of the Lyapunov exponents within the context is the size of the sample set used. The variance sample sets from this decoder are just too small for use with

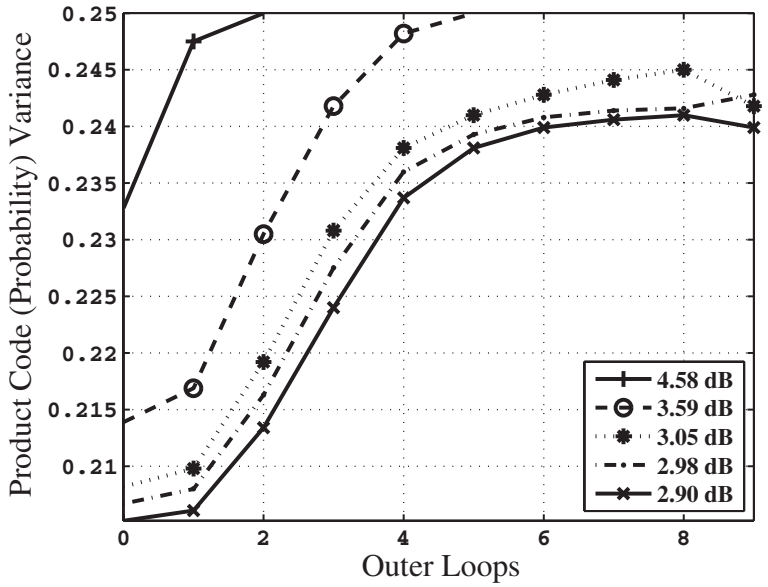

Figure 4.3: The convergent behaviour towards 0.25 of the the variance on a product codeword for successive outer loops.

Lyapunov exponents. Instead, two very simple additions are used with the variance test to aid in early exit detection. The first test is a test for increasing monotonicity. An increasing monotonic function is a function that only increases or stays constant, but never decreases. This will test whether the function is converging towards 0.25 or not. If the function ceases to be an increasing monotonic function, the decoder exits. However, a large spike in a decoding, which is still increasingly monotonic, can be indicative of an unstable decoding[1]. Thus, the final feature tests for variances larger than 0.25, as this can detect both unstable decoding spikes and decodings that drift beyond 0.25. These two tests can be used in conjunction with the variance thresholding technique for decoder exit control and early exit detection.

4.6 Optimised and Static decoders

As discussed earlier, the number of inner loops ω can have a dramatic effect on the decoding performance, both BER and time based. Too low an ω can introduce errors due to too short a belief-propagation stage. And too high an ω can create redundant processing, because the decoding may have already converged earlier in the decoding. However, selecting an ω which offers the best performance at a defined SNR is a challenging task. One way to do this would be to have some feedback system which would adjust the ω after an outer loop. A challenge arises in determining how to evaluate the decoding and determine which ω would best suit that product codeword at a specific SNR. This is not a trivial problem to solve. As such, another simpler approach has been chosen. This technique pre-evaluates a product code constructed

[1] A decoding that is oscillating between a few codewords

with specific component codes on a certain channel. The pre-evaluation determines which ω provides the best BER performance for the minimum redundant processing at a specific SNR. Also, because the optimised decoder is aware the SNR at which it can successfully decode, it can also determine at what SNR it cannot decode. Subsequently, the decoder can exit immediately from a decoding that will not succeed, thereby saving significant processing time. The optimised approach does have its limitations. Mainly, it requires the decoder to be fully characterised before it can be used. This is a sometimes infeasible task, especially if the system is dynamically shifting between different component coding techniques. Therefore, an alternative approach would be to define a static ω for all SNRs. The static system provides a somewhat naive approach, however, if an ω that offers good performance across all SNRs is found, this approach could be a valid alternative.

Although the optimised and static techniques use different approaches at the belief-propagation stage to selecting the value of ω, both techniques do share some similarities. Both techniques may use any of the presented methods for the codeword reliability estimation, the belief-aggregation and the exit test stages. However, for the optimised decoder a syndrome exit test is more appropriate. This is because the syndrome technique provides the most guaranteed exit determination. In using the syndrome exit test, the maximum number of iterations required would be known from pre-evaluation and the losses incurred by using a syndrome technique are understood. If it is deemed too costly, a more efficient technique could be used. Additionally, the need for dynamic early exit detection is irrelevant on the optimised decoder, which is aware of the SNRs at which it can decode. It is not inconceivable that for a well understood product code and component code combination the defined number of outer loops technique would not offer the best solution. Considering that the optimal number of outer loops is known for each SNR, the number of outer loops that offers the highest probability of a successful decoding could be used. Exit test techniques such as the variance threshold may offer slight decoding time performance improvements, these come at the possible cost of accurate decoding. The static techniques use the variance threshold, as the aforementioned techniques could run for an indeterminate length and the cost of a syndrome technique could become very high.

4.7 Chapter Summary

In this chapter a modification to the belief-propagation algorithm is presented. The modification allows for the parallel decoding of LDPC product codes to improve the decoding time performance. The decoder exploits the fact that all the component codes are encoded by i.i.d. encoding processes and as such can be independently decoded. To maximise the parallelism of the decoder, both the row and column

codewords are decoded simultaneously. The decoder adds three additional stages to the original belief-propagation algorithm: the codeword reliability estimation, the belief-aggregation and the exit test stages. The codeword reliability estimation and belief-aggregation stages are introduced to tackle the issue of evaluating and recombining multiple codewords, some of which contain unreliable information. As such, the codeword reliability estimation stage estimates the reliability of the information, so that during the belief-aggregation stage all the unreliable information can be removed from the final product code. The exit test stage is introduced in an attempt to reduce the computational complexity of calculating the syndrome for every codeword in the product codeword. The core belief-propagation stage has been modified such that it no longer controls when to exit from a decoding, but rather runs for a defined number of iterations for every single codeword decoding. This process introduces the question of what number of belief-propagation iterations need to be performed per one parallel belief-propagation iteration. The selection of the value is not trivial. Consequently, two solutions are presented. A static technique, which uses a static number of iterations regardless of channel conditions, and an optimised technique, which predetermines the ideal number of iterations for specific SNRs.

5 RESULTS AND ANALYSIS

The parallel belief-propagation decoder for product codes presented in this dissertation, see Chapter 4, offers a solution which could provide dramatic performance improvements, whilst still maintaining the decoding BER performance of the belief-propagation decoder. There are two key aspects to the decoders performance; the decoding BER performance and the parallel performance. The BER performance is tested in three parts; a comparison of the decoding BER performance using different internal components, the impact of ω on the system and finally a comparison to other existing algorithms. This quickly highlights what impact the additional components have had on the decoding BER performance. The decoder's parallel performance is then analysed to determine whether the degree of performance improvement offered by the parallelism justifies any losses incurred.

All the tests of the decoder use a $(1023, 781)$ LDPC code as the component coding technique [4]. All simulations were performed through Matlab [46]. Matlab does not present the ideal platform for performance testing, as it can be a somewhat cumbersome and slow testing platform. However, for the testing of a proof-of-concept design, as presented in this dissertation, it offers the shortest development time due to the numerous advanced functions available to the user. If this decoder were to be implemented on a real world system, a language such as C++ would be a more ideal platform as it is a higher performance language. The system used for parallel performance testing was an Ubuntu server running two quad core Xeon E5410 processors with 8GB of RAM, capable of running eight simultaneous threads. To ensure the performance results were valid, the server had no additional load and the server was headless to ensure the processor on the system was not being limited by background tasks. Obviously, it is impossible to ensure that the processor does not perform any other background tasks, but every effort was made to limit their impact.

At this point two metrics are defined that are used throughout the results section. The first metric is a generic measure of the amount of processing performed by a decoding. This metric is the processing product λ and is given by

$$\lambda = \omega \times \Omega, \tag{5.1}$$

where ω and Ω are the number of inner and outer loops, which have been defined in

Table 5.1: The legend for all the techniques used in the results section.

	Technique
AS-x	Unweighted Average using Syndrome exit test with $\omega = x$
WAS-x	Weighted Average using Syndrome Exit Test with $\omega = x$
AV-x	Unweighted Average using Variance threshold exit test with $\omega = x$
WAV-x	Weighted Average using Variance threshold exit test with $\omega = x$

earlier chapters. This value gives a system independent assessment of the processing performed by a decoding. This can be extended further to the redundant processing Λ given by

$$\Lambda = \frac{\lambda - \lambda_{min}}{\lambda_{min}} \tag{5.2}$$

where λ_{min} is the minimum processing product required to successfully decode a codeword under identical conditions. Redundant processing is the amount of processing performed above and beyond the absolute minimum required to successfully complete the decoding.

One of the challenges in this chapter is that there are multiple variations of the decoder. As such, a legend to identify the decoders is introduced. The legend is a simple concatenation of the different techniques used. An example variation would be **WAS - 1** which would be a Weighted Average using the Syndrome exit test where $\omega = 1$. The only codeword reliability estimation techniques that are implemented are the reliability estimate weighting and no weighting; as such the **W** indicates the use of the weighting technique, and an absence thereof indicates the use of the no weighting technique. The only belief-aggregation technique tested is the averaging technique; as such all techniques will contain the **A**. The final selection is the exit test used. The two exit tests implemented are the syndrome and variance threshold techniques; as such the final letter can either be an **S** or **V**. Table 5.1 shows a summary of the different combinations of techniques for the reader's reference.

5.1 BER Results

The decoding BER performance is one of the key metrics used in coding theory. It is defined as

$$\text{BER} = \frac{b_e}{b_s} \tag{5.3}$$

where b_e is the number of bit errors and b_s is the length of the bit stream [7]. These BER measurements are typically taken at worsening SNR levels on different trans-

mission channels. From a BER graph the behaviour of a technique under worsening channel conditions can clearly be seen. This is the standard means by which all coding techniques are evaluated. All points on the graphs were an average of at minimum three unique test. Due to the complexity and time requirement for a single decoding it was infeasible to perform more. However, within the waterfall region of the BER graphs it was sometimes necessary to perform more tests due to the instability of the decodings at this point. Thus, additional tests (between 20 and 50 tests) were run at these points to negate the unstable behaviour induced by the sharp waterfall regions.

5.1.1 Comparison of Different Presented Techniques

To evaluate the decoding BER performance of the parallel belief-propagation algorithm a few different sets of results are presented, which fully describe the decoder's performance. Initially, the internal behaviour of the system is investigated, looking at the impact on the BER for different combinations of codeword reliability estimation and belief-aggregation techniques. Then, the effectiveness of the exit tests is compared, looking at both the BER and early exit determination. The dependence of the decoding BER performance on the number of inner loops ω is then explored. The advantages offered by using the optimised decoder over the static decoder, see Section 4.6, are then considered. Finally, the decoder is compared to relevant existing techniques to evaluate whether the modifications to the system have resulted in a degradation of the decoding BER performance.

Codeword Reliability Estimation and Belief-Aggregation

The codeword reliability estimation and belief-aggregation stages work in conjunction. These two stages are responsible for identifying unreliable codewords and then reducing their impact on the decoding BER performance for the recombined product codeword P'. In Chapter 4 a few different techniques were suggested for both codeword reliability estimation and belief-aggregation. Subsequently, there are multiple different combinations of these techniques which can be tested. In this section the decoding BER performance for different combinations of these techniques is investigated.

There are two different codeword reliability estimation techniques that are being tested; a weighted and an unweighted average. These two techniques are combinations of the averaging belief-aggregation technique and the reliability estimate weighting, or the no weighting, techniques. Figure 5.1 shows the BER results for these two techniques. From the results it is evident that the weighted average offers a 0.07 dB improvement to the decoding BER performance compared to the plain average. Thus, the process of evaluating a codeword's reliability to minimise the

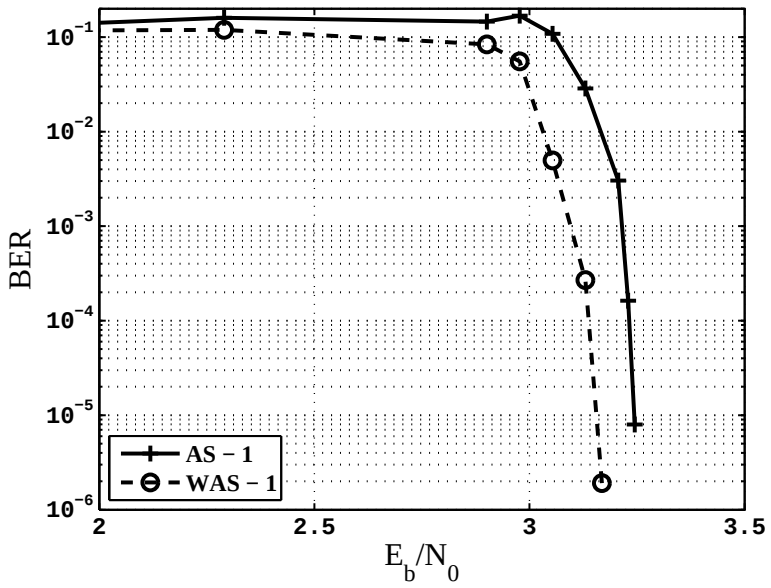

Figure 5.1: The impact of different codeword reliability estimation and belief-aggregation stages on the decoding BER performance.

impact of unreliable codewords on the product codeword has merits. Evidently, the lower weighting of the unreliable codewords during the averaging strengthens the overall codeword's reliability after recombination. As such, a more reliable codeword is utilised for following iterations, thereby further improving the performance. Similarly, if unreliable information is included in the recombined product codeword when using the unweighted average, that information only undermines the following iterations, thereby further degrading the decoding BER performance. This effect may be worsened if an inadequate exit test is used.

Exit Tests

The traditional exit test of $cH^T = \bar{0}$ can be very costly for long codes; see Section 4.5. In order to try to reduce the cost, a few alternative techniques were presented. The variance threshold technique is chosen for comparison, because of the predictable point of convergence for the variance on a decoded code, shown in Lemma 4.5.1. Figure 5.2 shows the results for the exit test testing. The techniques that were tested were a plain and weighted average using the syndrome exit test and a plain and weighted average using the variance exit test.

It is immediately evident that the use of the variance threshold technique has a negative impact on the performance. This would indicate that the variance threshold technique consistently exits slightly early, as a late exit will not affect the BER performance. The decoding BER performance at the waterfall region of the graph is degraded by an approximately 0.1 dB worsening of the decoding threshold. An

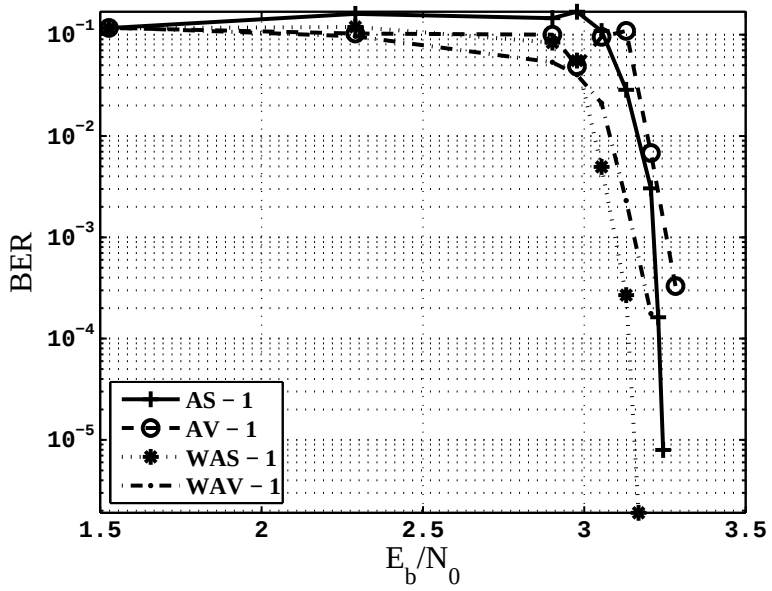

Figure 5.2: The impact of different exit tests on the decoding BER performance.

interesting feature for both the plain averages is a small dip in the decoding BER following the initial rise in the waterfall region. The results were consistent across multiple simulations, which suggests some peculiarity in either the decoding or the parity-check matrix used.

A minor digression from BER performance needs to be made at this stage as the exit test does not only effect the decoding BER performance. Another important feature of the alternative decoding techniques developed is the early detection of undecodable product codewords. Figure 5.3 shows the processing product for the same decoding techniques at worsening SNRs.

As would be expected, the techniques using the same exit test track one another closely. At high SNR values the syndrome exit tests exit earlier than the variance threshold tests. This is because the variance threshold tests require two successive tests to yield a stationary variance before an exit can occur. Subsequently, the variance threshold techniques require one additional decoding iteration for SNRs down to approximately 3.4 dB. At 3.4 dB the decoder (using the specified component coding technique) begins to reach its failure point. As such, the syndrome techniques start to show exponential increases in processing product as a greater number of component codewords become undecodable. The processing product λ continues to exponentially increase until the decoder's processing product saturates at a maximum λ of 20. At this threshold point the variance threshold technique exits early on a decodable decoding, resulting in the variance threshold exit tests having a higher decoding threshold. This is a flaw in the technique, as it could have offered similar

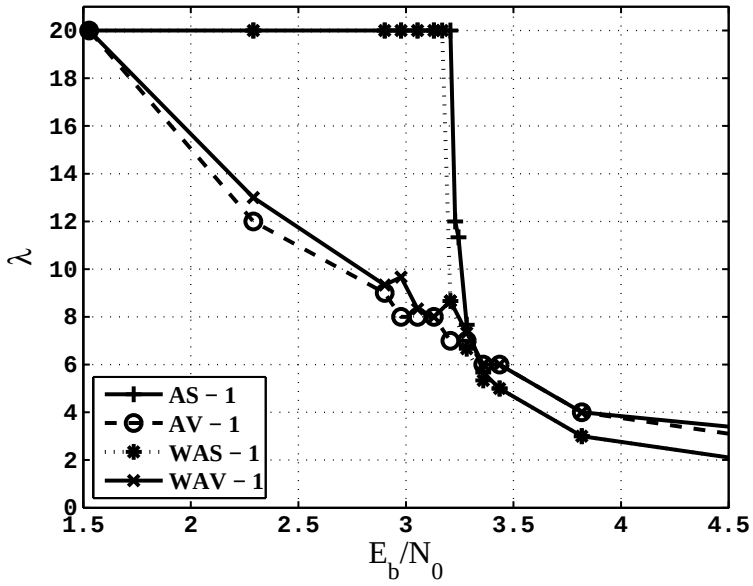

Figure 5.3: The impact of different exit tests on the processing product λ for a decoding.

performance to the syndrome technique. However, following that point the variance threshold continues to exit earlier than the syndrome technique, with a 60% earlier exit than the syndrome technique at 3.13 dB. This effectively means the variance threshold decoder needed less than half the amount of time used by the syndrome technique to reach the same conclusion, which is a significant advantage. The exit times for the variance techniques increase with lower SNRs, and eventually at 1.5 dB both techniques have the same processing product. Although, the variance threshold technique does show early exiting for SNRs soon after the threshold point, eventually it offers no advantage compared to the syndrome technique. This is a flaw that needs to be addressed by either the addition of additional small tests (like those described in Section 4.5) to refine the variance threshold technique's behaviour, or an entire rework of the technique.

Realistically, if the system were to be operating on some real world channel, the operational SNR on that channel would fluctuate only marginally below the decoding threshold, as the channel coding technique would be chosen to suit the specific channel conditions. For the AWGN channel the likelihood of the SNR extending beyond the operational region, to a point where the variance threshold technique does not exit early, would be very small, so the likelihood of a failed early exit would be limited. Therefore very few decodings which exceed the threshold by such a margin would be expected. Thus, the variance threshold's successful early exit point is a tolerance taken into account within a design. The next factor that can have a noticeable impact on the decoder's performance is the choice of ω, as it defines the

amount of belief-propagation performed before a pseudo-*posteriori* is created.

5.1.2 Impact of Higher ω Values

The choice of ω has a direct impact on the performance of the decoder. If ω is chosen poorly it can negatively affect the decoding BER performance. If ω is chosen too low, insufficient belief-propagation iterations could be performed and a stable state for the Bayesian network may not yet be inferred. Therefore, at the end of the belief-propagation iterations when the pseudo-hard-decision is made, it would be a poor decision and the subsequent decodings are weakened. Figure 5.4 shows the decoding BER at different SNRs for WAS decoders.

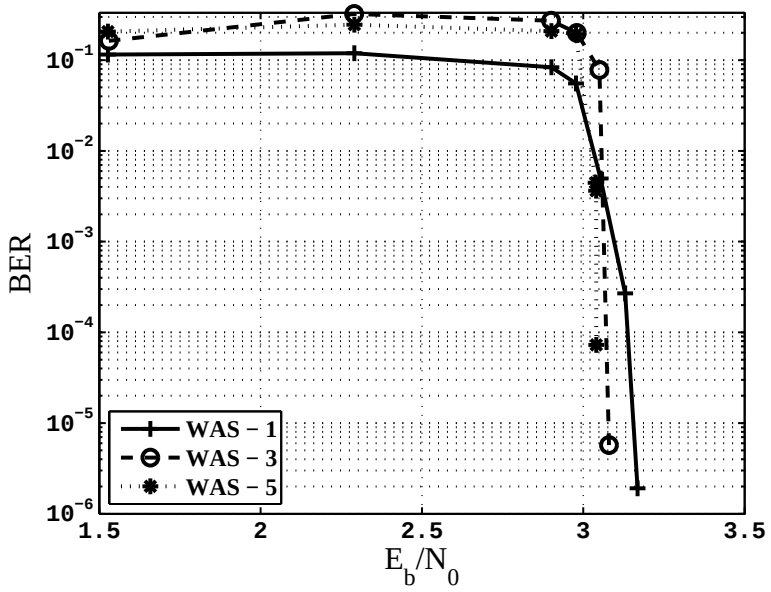

Figure 5.4: The impact of different ω values on the decoding BER performance.

The three ω values used in the testing are one, three and five. From the graph, as would be expected, the ω of one offers the worst performance. Evidently, for the component code and channel used, an ω of one is an insufficient number of belief-propagation iterations for the decoder to infer a good marginal distribution across the Bayesian network. With an ω of one the performance is poorer than when using values of three or five. When ω is set at three the performance increases significantly, with a 0.08 dB gain for two additional iterations, thereby showing that even a small increase in ω can offer good returns. However, when using an ω of five (again an increase of two iterations) the decoder only shows a 0.04 dB improvement. This is a 50% reduction in efficacy compared to the increase from one to three. However, this is what would be expected as improvements to the performance by the addition of more belief-propagation iterations are limited. Therefore, as the maximum ω is approached the advantage offered by the increasing ω quickly deteriorates.

5.1.3 Optimised vs Static Decoder

As discussed in Chapter 4 an optimised decoder which has had the ω values pre-calculated could offer significant decoding time and decoding BER performance advantages. The optimised decoder uses an ω value which has been predetermined for a specific SNR and provides a high likelihood of successful decoding. As such, not only does the optimised decoder decode successfully, but it also performs the decoding in the shortest time possible. The obvious limitation of an optimised decoder is that it cannot be used on just any channel using just any component code; the optimisation is only valid under specific criteria. The predetermined behaviour of the optimised decoder would purport that the optimised decoder would outperform the static decoders already previously outlined. In this example, the optimised decoder used a maximum ω of 20 and a weighted average combination with the syndrome exit test. As such, for comparative reference the optimised decoder will be compared to the weighted average techniques using different ω values. Figure 5.5 shows the comparative performance of the optimised decoder compared to its static counterparts.

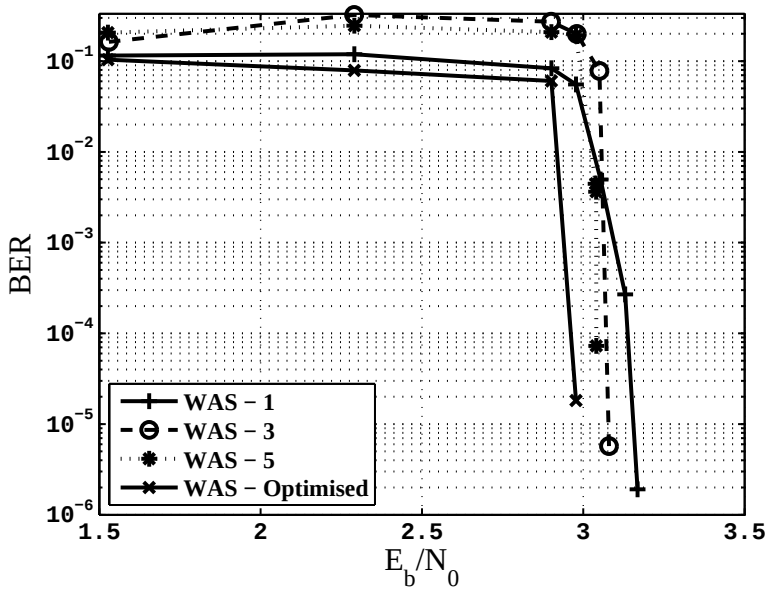

Figure 5.5: The decoding BER performance of the optimised decoder compared with its static equivalents.

The optimised decoder shows a marked performance advantage across all the tested SNRs. At the decoding threshold the optimised decoder shows an approximate improvement of 0.06 dB over the static WAS-5 technique. Were a static decoder with an ω of 20 compared to the decoding threshold of the optimised decoder, the results would be similar, because then both decoders propagate beliefs for the same period before the pseudo-hard-decision is made. However, using an ω of 20 would

result in significant redundant processing at higher SNRs. So the choice of the ω in a static decoder becomes somewhat of a trade-off between BER and time performance. An ω of 5 offers a good trade-off between BER and time performance for this channel. This comparison is somewhat meaningless however, as no reference to relevant other techniques is made. The only way to properly ascertain the performance of the decoder would be to compare it to other relevant techniques.

5.1.4 Comparison to Existing Techniques

To properly determine the decoding BER performance of the presented decoder, it is necessary to compare these techniques to relevant other decoders. Due to the decoding time necessary for the decoding of an LDPC product code, no real work has been completed in this field [47]. The techniques that are applicable to product codes, such as the Chase-Pyndiah decoder, would have impractical decoding times for LDPC product codes [17, 22]. However, a few base techniques have been devised to indicate comparative performance: uncoded BPSK, plain belief-propagation decoding, concatenate belief-propagation decoding and iterative concatenate belief-propagation decoding. All these techniques can perform up to the same number of belief-propagation iterations. The limitations on the number of iterations for each technique ensure that a true comparison is made.

Uncoded BPSK This technique needs very little description. It is the BER performance the system can achieve when using only a modulation technique, in this case a BPSK modulator.

Plain Belief-Propagation As the name suggests, this technique simply performs a single belief-propagation decoding for row codewords only. It can perform up to a maximum of 120 iterations per codeword before exiting. This gives an indication of the performance that could be offered by simply using the basic belief-propagation decoding algorithm. No information is shared between codewords.

Concatenate Belief-Propagation This technique introduces the concept of concatenating the row and column codeword decodings to achieve better performance. This concatenate process is what is used in turbo codes as information from one decoding is used as extrinsic information in the next decoding. This chaining of decodings should result in an improvement over the standard algorithm. For this technique a maximum of 60 iterations can be performed for decoding each codeword in both the dimensions.

Iterative Concatenate Belief-Propagation One limitation of the concatenate LDPC decoder is that it does not reuse the information in successive decodings to further enhance the performance. The iterative concatenate LDPC decoder does

this by iterating the concatenate decoders multiple times, always feeding the output from the last decoding into the next. Using this approach, the greatest performance can be achieved as the beliefs have opportunity to propagate throughout the entire product codeword. This algorithm will share all the information available across all the codewords to facilitate a successful decoding. For this technique a maximum of 20 belief-propagation iterations are performed per codeword in each dimension to a maximum of three outer iterations[1]. This technique should offer the best performance possible, as it makes no performance-limiting accommodations in its iterative process.

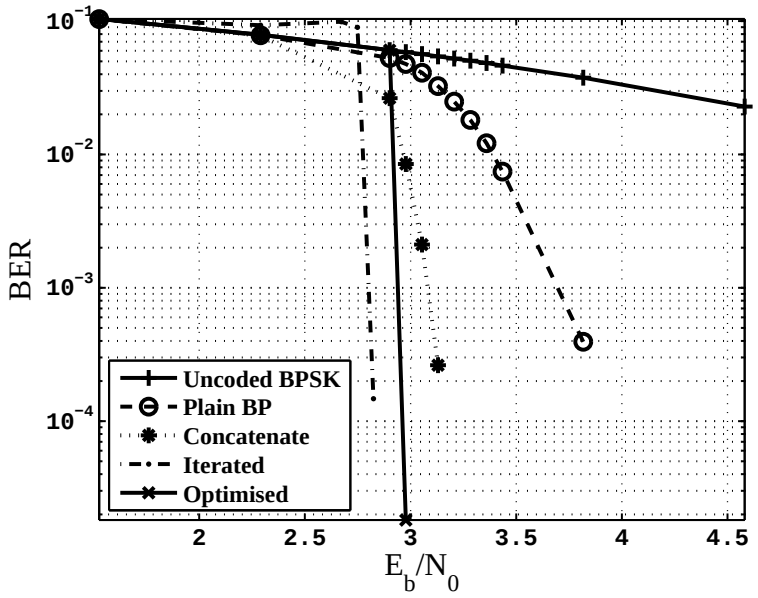

Figure 5.6: The decoding BER performance for the optimised decoder compared with existing algorithms.

From the comparative performance - results seen in Figure 5.6 - it is evident that the process of concatenating and iterating the decoding process increases the performance substantially. By concatenating the belief-propagation algorithm and decoding the rows and then columns, a 0.69 dB performance improvement over the plain belief-propagation algorithm can be achieved. Although both techniques use an equal number of belief-propagation iterations, the concatenate technique uses information sharing between the two decodings. As such, row decodings, which could not be successfully decoded, have their errors corrected during the column decodings. This then improves the overall performance, because information is shared between the two decoding processes. This is fundamentally the same process when the concatenate process is iterated. However, in the iterated concatenated belief-propagation algorithm the information is shared throughout the product codeword. The code-

[1] An outer iteration is a shortened concatenate decoding

words which are undecodable during the first round of row and column decodings have a better chance of being successfully decoded during the second iteration, similarly for the third iteration. As such, the use of the iterative concatenate technique offers a further 0.31 dB improvement over the concatenate belief-propagation technique.

The comparative performance of these techniques with the proposed decoder is of critical importance. The result will show whether the additions to parallelise the decoder have severely hampered the decoding BER performance. To provide a baseline, the serial algorithm will be compared to the optimised decoder. The optimised decoder, as already shown, outperforms the WAS-5 static decoder by 0.06 dB. However, the optimised decoder suffers a 0.16 dB worsening when compared to the iterative concatenate technique. This is likely due to the belief-aggregation and codeword reliability estimation stages. The unreliable information remains too prevalent in the resulting product codeword and, as such, weakens the decoding BER performance. However, although the parallel decoders show a worsening in the decoding BER performance, it is not so significant a loss that the advantages offered by the parallel decoding do not outweigh these losses. In fact, it is likely that the decoding time performance improvement is more significant, because the decoding BER performance can be improved by simply using a more powerful LDPC code.

5.2 Parallel Performance Results

The decoding BER performance is not the only key performance indicator of the parallel decoder. In this work the aim is to improve the decoding time performance of LDPC product codes through the use of parallelism. As such, it is necessary to quantify the decoding time performance benefits offered by the parallel decoders. Four features of the decoder are investigated to ascertain the decoding time performance: the implications of ω on the decoding performance, the effect of different exit tests on performance, the redundant processing incurred through the use of an overly large ω and the real world performance of the decoder on a parallel system.

5.2.1 Performance Dependence on ω

As discussed previously in Section 5.1, the use of too large or too small an ω can have an impact on the decoding BER performance. The decoding time performance also shows a dependence on the value of ω. The parallelism of this decoder centres around the belief-propagation decoding section of the decoder. A series of belief-propagation iterations are performed in parallel for all the component codewords in the product codeword. However, for each of these belief-propagation stages there are some additional serial processes[2] which facilitate the parallel decoding. If too

[2]The codeword reliability estimation, belief aggregation and exit tests.

few belief-propagation iterations are performed, the serial section of the algorithm increases. If one considers the implication of Amdahl's Law, a larger serial section within the decoder would have a negative impact on the theoretical maximum *speedup*. Table 5.2 shows the percentage of the decoder that is parallel for an increasing ω.

Table 5.2: The percentage of the decoder that is parallel for increasing ω.

Inner Loops (ω)	Percentage of the decoder that is parallel
1	0.9633
2	0.9807
3	0.9862
4	0.9905
5	0.9923
7	0.9943
10	0.9960

When the data from Table 5.2 is shown on a graph, seen in Figure 5.7, a convergent trend is observed. Intuitively this makes sense, as an increase in the number of iterations reduces the serial section of the decoder until it is negligible. If one fits a curve to this data using a power curve fit with a curve fitting tool the equation

$$f(x) = -0.03784\omega^{-0.885} + 1.001,$$

is fitted to the data points. Solving for the point that the decoder is negligibly serial, $f(x) = 1$, it is found that an ω of 60.74 would be needed for the decoder to be theoretically 100% parallel[3]. In actuality, the decoder can never really be 100% parallel as the communication costs between parallel processors can never be removed. At an ω of 60.74, the serial section of the decoder becomes negligible in comparison to the parallel section. With an ω of 60.74 the decoder can theoretically achieve its maximum *speedup*. Logic would dictate that then using an ω of 60.74 would result in the decoder offering its best performance. However, this does not account for the actual number of iterations required to achieve a successful decoding at any given channel SNR.

5.2.2 Redundant Processing Trade-off

Redundant processing Λ was defined at the beginning of this chapter as the amount of processing required for a decoding above and beyond the minimum possible processing for that decoding. Λ provides a percentage measure which can be used to indicate

[3]effectively an ω of 61, as part iterations cannot be performed.

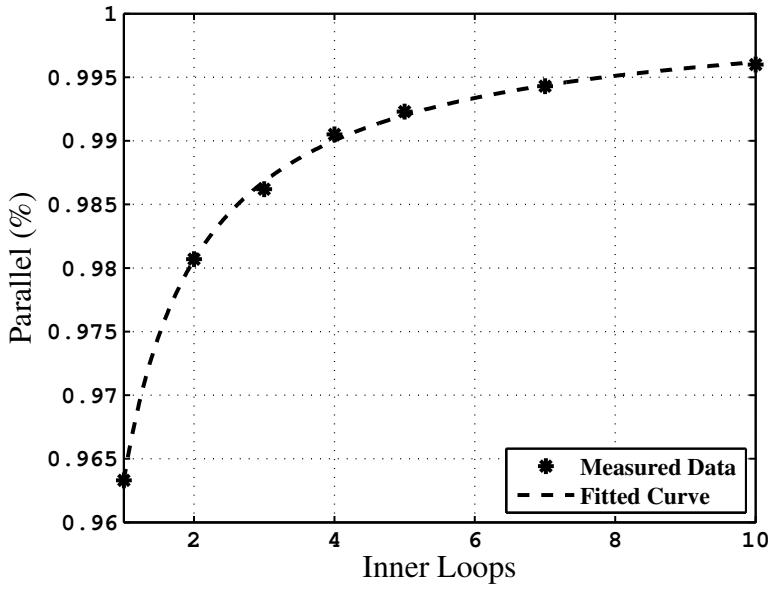

Figure 5.7: The percentage of the decoder that is parallel versus ω at an E_b/N_0 of 3.8 dB.

how much more processing a decoding performs above the minimum necessary. This essentially relates back to the number of inner loops, or belief-propagation iterations, within one decoding. Using the data from Table 5.2 to calculate the *speedup* predicted by Amdahl's Law [34] and Table 5.3 to determine the redundant processing, an estimate of the expected performance benefit can be determined.

Table 5.3: The processing product λ for increasing ω.

Inner Loops (ω)	Processing product (λ)
1	5
2	6
3	9
4	12
5	15
7	21
10	30

The performance benefit is the net difference between the *speedup* offered by increasing ω and the redundant processing increase by doing the same. These two features will reach some equilibrium point for a specific SNR which offers the best performance.

Figure 5.8 shows the *speedup* S predicted by Amdahl's Law, the *speedup* loss from the redundant processing Λ and the net *speedup* for the decoder for a system tested on an AWGN channel at 3.8 dB. Firstly, as would be expected, the increase in ω results in an increase in parallel performance, with the *speedup* tending towards 10. Similarly,

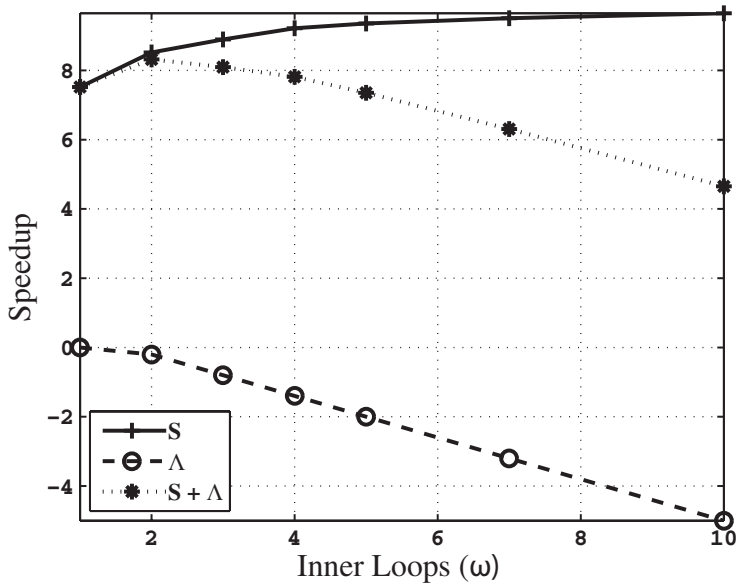

Figure 5.8: The performance trade-off between *speedup* and redundant processing for changing ω at an E_b/N_0 of 3.8 dB on a ten processor parallel system.

the redundant processing increases with an increased ω, resulting in an increasingly negative *speedup*. If the *speedup* gains predicted by Amdahl's Law and the *speedup* losses from the redundant processing are combined, one observes the difference curve that shows a maximum *speedup* when ω is two. Due to the decaying nature of the speedup, the redundant processing quickly dominates the performance. It is evident that the advantages gained by using a larger ω are easily overwhelmed by the losses from the redundant processing. This result clearly shows the naivety in a decision to increase ω to offer greater performance. The maximum speedup predicted at ω is two does not stand for all conditions. In fact, it only offers the best trade-off for the given component code on an AWGN channel at 3.8 dB. This analysis would need to be performed again for each SNR to get an indication of real world performance.

5.2.3 Performance Advantage of the Alternative Exit Test

Throughout the discussion around the exit tests, it has been assumed that the use of a variance threshold offers performance benefits over the traditional syndrome technique. The variance threshold technique should offer better performance as it needs to perform less processing in order to correctly determine whether a codeword is successfully decoded. Table 5.4 shows the processing time taken for each of the two exit test techniques.

From these results it is clear that the variance threshold does offer a performance benefit. The variance threshold performs the exit test in 2% of the time it took to perform the syndrome exit test. In itself, that appears to be a significant improvement, however, the performance improvement is relative to the parallel section of the

Table 5.4: The time required to perform the different exit tests for an ω of one.

Exit Test	Time (s)	Percentage of Total (%)
Syndrome	0.307	0.133
Variance Threshold	0.006	0.0027

decoder. The section of the decoder which is parallel for an ω of one is 96.33% for the WAS technique. Considering that the variance threshold uses 2% of the time of the syndrome technique, the effective parallel section of the decoder would be 96.46%. For a 10 core parallel processor, that equates to a 2.9% improvement in the parallel *speedup*, predicted using Amdahl's Law [14, 34]. This may seem insignificant, but when one is working with many processors such an improvement can become substantial. For a 10000 core parallel processor this would result in a 3.7% improvement in the parallel *speedup*. Although this value is small, if one was working with a component codeword that has a shorter serial decoding time, these small savings can become significant. Furthermore, on systems that have very performance sensitive transmissions the small time saved may be of critical importance. For significantly high ω the choice of exit test becomes somewhat redundant as the belief-propagation section of the decoder dominates the processing time.

5.2.4 Real World Performance

All the performance evaluations up to this point have been based around the theoretical evaluation of the decoder's performance derived from measured quantities. To ratify the assessment made already, it is necessary to analyse the decoder's real world performance. The decoder's decoding time was measured for an increasing number of parallel processors. The tests were performed using a WAS-5 static technique with an E_b/N_0 of 3.8 dB. Figure 5.9 shows the decoding time results.

Clearly, the decoding time shows an inverse proportionality to the number of processors used. There is a slight discontinuity seen at the transition from four to five cores. This discontinuity is likely caused by poor load management across the two processors during the transition between processors, which could be indicative of an issue in Matlab's multi-processor handling. As a result, there is a performance bottleneck when transitioning to the second processor, which is likely caused by equal distribution of load across the cores rather than across the processors. For a single cored system a decoding time in excess of 6×10^4s was measured. When using all the cores, the decoding time using all the processors was less then 1×10^4s. If it is assumed that the single core decoder is the fastest serial decoding, then Equation (2.4) can be applied. If so, the eight core parallel decoder results in a 7.26 *speedup* over the serial decoder. This is a *speedup* of 91% of the theoretical maximum of eight. This shows that the decoder does indeed offer good parallel performance even on systems with a relatively small number of processors. As a result, it can be inferred

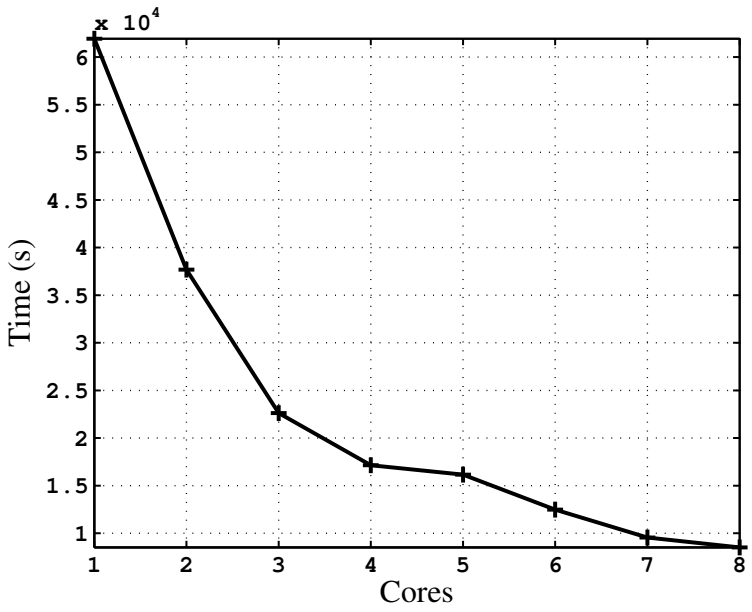

Figure 5.9: The decoding time on a multi-cored processor using the WAS-5 static technique at an E_b/N_0 of 3.8 dB.

that for a large number of parallel processors the parallel performance increase could be equally impressive.

5.3 Chapter Summary

In this chapter, the performance of the parallel belief-propagation decoder was tested. Both the decoder's decoding BER performance and parallel performance are tested. The BER testing was spread across multiple tests, investigating the impact of each of the components added to the original belief-propagation algorithm. It was found that the use of a codeword reliability estimation technique did provide a 0.04 dB enhancement in performance when recombining the product codes compared to an unweighted recombination. The alternative means of exiting a decoding, the variance threshold test, was shown to provide an approximately 0.1 dB worsening of performance when compared to the traditional syndrome exit test. However, the losses were made up by a 60% earlier exit for undecodable codewords over the syndrome technique. It is also seen that the choice of ω can have a dramatic effect on the performance of the system. When compared to existing solutions the parallel belief-propagator showed a 0.16 dB worsening in performance compared to the iterative concatenate technique. This is likely a result of a non-ideal belief-aggregator leaving unreliable information in the final product codeword. Although, there was a worsening in performance, it was not without equally impressive decoding time performance gains. The decoder was analysed and tested to determine its parallel performance. The tests were both theoretical and practical. It was determined that for an ω of 60.74 the system would be essentially parallel and no further improvements could be

gained by increasing ω. It is also shown that the performance could also be worsened by increasing ω as it would result in redundant processing being performed. It is shown that for an SNR of 3.8 dB an ω of two offered the best performance. To ratify all the performance results a real world test was performed and it is found that an eight core system could offer a 7.26 times *speedup* over its serial counterpart. This is 91% of the theoretical maximum. In conclusion, the slight worsening in decoding BER performance results in a small trade-off in terms of the decoding BER performance. The decoder is thus deemed successful although it worsened the decoding BER performance. To improve the BER performance all that is required is a more powerful LDPC code. However, to reach the same kind of decoding time performance a significantly shorter code with a much weaker decoding BER performance would be needed.

6 CONCLUSION

In this dissertation a design of a parallel belief-propagation decoder for LDPC product codes has been investigated. The square structure of a product code, containing multiple independent codewords, makes the decoding a very costly process. The designs in this dissertation relied on the fact that the component codewords within the product codeword are independently encoded by i.i.d. encoders. As a result of this, each codeword in the product codeword can be independently decoded. The independence of the decodings allows the algorithm to be parallelised for enhanced decoding time performance.

Two unique solutions were designed around the independence of the component codewords. The first technique is a hard-decision erasure decoding algorithm, which uses a graph theoretic approach to parallelise the decoding. The graph structure can then be used to identify decodable codewords within the product codeword, and only process those which are decodable. Additionally, the algorithm suggested that it would identify the codewords that, when decoded, would correct the highest number of erasures. This introduced the problem of how to identify the ideal codewords for decoding. It was discovered that there is no simple or efficient way to solve this problem and the only way is through an exhaustive search. Without the novelty of an efficient decoding, it was deemed that this algorithm provided limited advantage and as such was never implemented. The next decoder made use of lessons learned in the design of the graph-based decoder.

6.1 A Parallel Product Code Decoder

The next design moved away from attempting an efficient decoding and rather focused on performing all the decodings simultaneously. This decoder targets the decoding of LDPC product codes using a soft-decision belief-propagation decoding algorithm. The modified belief-propagation decoding algorithm introduced three new additional stages to the belief-propagation algorithm to facilitate the decoding of a product code. To improve the performance, all the component codes from all the dimensions are processed simultaneously. To expedite this, the codeword reliability estimation stage and belief-aggregation stage were introduced. The codeword reliability estimation stage identifies unreliable codewords and, using a weighting vector, reduces their impact when all the codewords are recombined during the belief-aggregation stage. The recombined product codeword then needed to be tested to see if it is valid. The traditional exit test of $cH^T = \bar{0}$ for every codeword can be

very costly for a code the size of an LDPC product code. To tackle this problem, a few new techniques were introduced, which moved the exit test away from the belief-propagation algorithm and rather used techniques which are applied to the entire product codeword. Other than moving the exit test from the individual codeword's decodings, the belief-propagation algorithm was modified in another way. To facilitate the 'lock-step' behaviour of some parallel processors, such as GPUs, the belief-propagation stage of the decoder was made to run for a defined number of iterations. It was expected that these extensions would result in a degradation of the decoding BER performance, but this would have come at an increase in the decoding time performance.

6.2 Implications of Parallelism on BER Performance

The decoding BER performance is of critical importance to any telecommunication system. It was observed that for the parallel decoder, the BER performance was affected by the different configurations of the decoder. Using the reliability estimate weighting as the codeword reliability estimation technique from Section 4.2, with a weighted average belief-aggregator from Section 4.4, a decoding BER performance improvement of 0.07 dB compared to the plain average is observed. Furthermore, the variance threshold exit test, which was the alternative exit test developed, is shown to offer a 0.1 dB worsening of the decoding BER performance compared to the traditional syndrome exit test. However, the variance threshold technique offered a 60% earlier exit than the traditional syndrome technique at 3.4 dB. This is a significant advantage, because processing time would otherwise be wasted on a decoding that would inevitably fail. The decoder's performance is shown to be severely impacted by the choice of the number of belief-propagation iterations (ω). For an increase of ω from one iteration to three iterations a 0.08 dB improvement is shown. An additional two iterations, thus five belief-propagation iterations in total, shows a 0.04 dB improvement. The diminishing returns are to be expected, as the belief-propagation algorithm begins to approach its bound for the given component codes.

The most important feature of the decoding BER results is the performance relative to other existing techniques. Going from a plain belief-propagation algorithm to a concatenate equivalent offers a 0.69 dB gain. Additionally, modifying that into an iterative concatenate scheme offers a further 0.31 dB improvement. When the iterative concatenate scheme is compared to the parallel decoder it is seen that the parallel decoder offers a 0.16 dB worsening in performance. It is postulated that this worsening is as a result of insufficient reduction of the unreliable codeword's information during the belief-aggregation stage. It is likely that a technique that improves the belief-aggregation stage could approach the iterative concatenate technique's performance.

6.3 Advantages of Parallel Design

All the modifications made are for the sole purpose of parallelising the decoding process to increase the decoding time performance. Amdahl's Law states that an algorithm's theoretical maximum *speedup* is directly related to the percentage of the decoder which is processed in parallel. It was shown that the percentage of the decoder that is parallel is directly related to the number of belief-propagation iterations ω performed within one iteration of the parallel decoder. The parallelism of the decoder shows a decaying behaviour. Through the use of a curve fitting tool it is found that for 60.74 belief-propagation iterations, the decoder effectively became completely parallel. Through the introduction of a new quantity, called redundant processing, it is shown that although performance is gained by increasing the number of belief-propagation iterations, the amount of wasted processing is also increased. For an SNR of 3.8 dB, two belief-propagation iterations offered the best *speedup* to redundant processing trade-off and would offer the greatest net *speedup*. However, all this analysis is largely theoretical in nature and a real world test was used to show the system's actual performance. The decoder shows a 7.26 times *speedup* over its serial counterpart on an eight core system. That is 91% of the theoretical maximum *speedup* of eight for an eight core processor.

6.4 Research Aims

The research had the aim of increasing the decoding time performance for the decoding of an LDPC product codes, whilst still offering performance comparable to similar techniques. The decoder shows a 7.26 times improvement in decoding time performance on an 8 core processor. This is a trade-off on the decoding BER performance, resulting in a 0.16 dB loss in performance compared to the best serial decoder. As such, the research is deemed to have successfully met its stipulated aims.

6.5 Further Work

The work presented in this dissertation is only a prototype and proof of concept. As such, there is significant room for improvement. During the design of the decoder, a few additional techniques were proposed which never went to prototype. These techniques could replace some of the techniques used in the codeword reliability estimation, belief-aggregation and exit test stages in the proposed decoder. In addition, a novel application of the work is also presented.

6.5.1 Codeword Reliability Estimation

Only one additional technique was considered for the codeword reliability estimation stage, however, this does not necessarily mean that there are not more additional

techniques. The convergence rate weighting technique uses a different metric to estimate the reliability of a decoding technique compared to the reliability estimation weighting. Rather than using the reliability factors, the convergence rate weighting technique generates a weighting factor based on the rate of convergence. It assumes that a high rate of change between outer loops is indicative of a good decoding. This technique requires two versions of the P to be stored, from a time instance before and after Ω decoding iterations. Therefore, the horizontal codewords' weighting factors are given by

$$\phi_i = \frac{1}{n_h} \sum_{j=1}^{n_h} (c_{j,i}^{\tau} - c_{j,i}^{\tau+\Omega}), \tag{6.1}$$

where τ is a *pseudo*-time variable indicating a stationary iteration, which is used as a reference point for the difference calculation. Similarly, the weighting factor for the vertical direction would be given by

$$\phi_j = \frac{1}{n_v} \sum_{i=1}^{n_v} (c_{i,j}^{\tau} - c_{i,j}^{\tau+\Omega}). \tag{6.2}$$

However, this technique is expected to have flaws. The first and greatest shortcoming of this technique is that a codeword which has been successfully decoded will show very little change between decoding rounds. Subsequently, it would have a poor weighting factor assigned to it. Thus, these successfully decoded codewords will have limited influence on the decoding, with codewords which are undergoing large changes being favoured in their stead. Although a high degree of change between iterations can indicate a convergence to a codeword, it could also indicate an unstable decoding. An unstable decoding takes place where the Bayesian network cannot be processed to a point where all the conditionals are in agreement. Consequently, the decoding can oscillate between many different codewords with massive changes between iterations. An additional issue with the convergence rate weighting as a codeword reliability estimation technique is that it requires two versions of the product codeword to be stored, which requires additional memory space. This can become problematic on systems with limited memory available. Lastly, no weighting can be generated for the first outer loop, because there is no previous product codeword to use as reference. Thus, no weightings can be generated for the first round. This would stall the decoding process as the first round would have to combine unreliable and reliable codewords together equally. As a result of these flaws this technique was not taken to prototype. Although, the convergence weighting technique is likely to offer better weightings during the transitionary stages. A possible solution to its weaknesses at the beginning and end of the decoding would be to hybridise it with the reliability

estimate weighting technique. The combination of techniques would likely offer good performance benefits.

6.5.2 Belief-Aggregation

Two additional belief-aggregation techniques are suggested which could offer improved performance by negating the effect of unreliable codewords. The two techniques are a most-likely symbol selection and voting technique.

Most-Likely Symbol Selection

The most-likely symbol selection technique differs from the averaging techniques in that it tries to minimise the amount of unreliable information in the product codeword. Rather than combining the probabilities from P and P^T, this technique only uses the most reliable symbol from either P or P^T. By taking only the most reliable information forward, arguably the most accurate product codeword undergoes further processing. Algorithm 6.1 presents pseudo code which describes the process involved in the most-likely symbol selection technique. The product codewords P and P^T are initially weighted by their respective weighting matrices. The weighted P and P^T matrices are then compared at each position in the product codewords. The higher of the two values after weighting indicates the most likely symbol for that position in the product codeword. As an example, if $\beta^h_{1,5} \times P_{1,5}$ was higher than $\beta^v_{1,5} \times P^T_{1,5}$, then for the position of $P'_{1,5}$ the conditional probability would be $P_{1,5}$, whereas if $\beta^v_{1,5} \times P^T_{1,5}$ is higher the conditional probability from the matrix would be $P^T_{1,5}$.

Algorithm 6.1 Pseudo-code describing the operation of the most-likely symbol selection technique.

Inputs: Row and column product codewords (P and P^T),
Weighting Matrices (β^h and β^v)
Outputs: Recombined product codeword (P')

$P = P \times \beta^h$ ▷ *Multiplications are performed element-wise*
$P^T = P \times \beta^v$ ▷ *Multiplications are performed element-wise*

for $i = 1 \rightarrow n_v$ **do**
  for $i = j \rightarrow n_v$ **do**
    if $P(i,j) \geq P^T(i,j)$ **then** $P'(i,j) = P(i,j) \div \beta^h_{i,j}$
    else if $P(i,j) \leq P^T(i,j)$ **then** $P'(i,j) = P^T(i,j) \div \beta^v_{i,j}$
    else $P'(i,j) = P(i,j) \div \beta^h_{i,j}$
    end if
  end for
end for

By ignoring the more unreliable information the product codeword should show both a better decoding time and BER performance when compared to that of the averaging techniques. One weakness would be that a false positive codeword could be chosen and hence weaken the quality of the conditionals within the Bayesian network. Two versions of the most-likely symbol selection technique can be used, either weighted or unweighted.

Unweighted The unweighted technique sets the weighting matrices discussed earlier to one. Then the selection is purely based on a comparison of the conditional probabilities between P and P^T. This technique is somewhat naive, as it ignores the reliability of the remainder of the codeword from which the most-likely symbol is being selected.

Weighted The weighted technique uses the weighting matrices β_h and β_v to weight the conditional probabilities accordingly. This should give a more accurate selection, as the reliability of the codeword is also taken into account.

The most-likely symbol selection technique could offer performance improvements over the averaging techniques presented. The most-likely symbol selection technique does not suffer from the weakness of the averaging techniques, because no unreliable information is included in the final product codeword. As such, it is likely that this technique could approach the performance of the iterative concatenate belief-propagation algorithm presented.

Voting

The second technique aims to use the two product codewords in a voting system to improve the decoding BER performance. The motivation behind a voting system is that if there are two very positive conditionals which are in agreement for a position in the product codeword, the resulting reliability should be more positive when combined than either of the two conditionals. Because, there are two independent and unrelated information sources in agreement the likelihood of the correctness of that information is increased.

After the weighting factors are applied and correctly scaled, rather than simply choosing or averaging the two values the information is used in a vote. The premise is that if both product codewords at a specific element have conditionals that are in agreement, for example 0.9 and 0.85, the resulting output should be even more positive than their combination. If the averaging or most-likely symbol selection techniques were used, the value of the resulting conditional would be 0.875 or 0.9 respectively. However, it is evident to an observer that the result is actually more

reliable due to both codewords 'voting' in agreement. Subsequently, both of the resulting values are in agreement and the system would vote with a higher probability towards that agreed upon conditional, rather than towards either of the values involved in the vote. This technique was not developed further due to the complexities involved in developing such a voting system. However, it could offer a marginal BER performance improvement over the iterative concatenate technique.

6.5.3 Exit Test

An additional exit test which uses a Kullback-Liebler Distance (KLD) threshold instead of the variance threshold presented in Section 4.5, is discussed here. The KLD, or divergence, is a measure of the similarity between two probability distributions [48]. This metric can be used in a similar fashion to the variance threshold approach presented previously. Since the KLD gives a measure of the similarity between two probability distributions, it can be used to measure the difference between two product codewords. The product codewords to be tested would be P and P', from before and after a decoding outer loop. If a high similarity is found, this could be indicative of a successful decoding. This can either mean the decoding has been successful and converged to the correct product codeword or it has converged to the incorrect product codeword. The KLD is defined as

$$d = \sum_{\gamma} log_2(\frac{f_\gamma}{g_\gamma}), \tag{6.3}$$

where γ is the length of the distributions and f_γ and g_γ are the probability distributions being compared. The KLD is not a true metric, as it is dependent on the direction of comparison. However, if the test remains constant it can be used to measure a convergent behaviour in the decoding. As before, the KLD on its own is insufficient to detect an early exit. However, the addition of similar helper tests would allow for early detection. This technique was not implemented in the main work due to its similarities to the variance threshold technique. It is unlikely that this technique would offer significantly different performance to that of the variance threshold technique.

6.5.4 Application

The last suggested route of future research looks more to the applications of this work. It is evident that this system could be implemented on any parallel processor available and can provide a significant performance advantage for the transmission of such large codes. One area in which such a decoder could offer significant advantage is the transmission across a wireless channel where a successful transmit can be challenging to achieve and the performance is degraded by the need for retrans-

mits. An LDPC product code could be transmitted across a harsh wireless channel and still be recoverable at the receiver. The presented decoder is applicable to large devices with the processing power to perform the decoding which need to transmit data across a wireless channel. However, the processing requirement for this decoder is too large for smaller devices like smartphones. Nonetheless, these devices could gain significant benefits from such a decoder. Due to the mobility of smartphones, they are often within communication distance of a host of other smartphone devices (at a public space or in the workplace). These devices are largely sitting idle and not utilising their full processing potential. The smartphone that needs to perform such a large task, like the decoding in this dissertation, would therefore utilise the processing power of other nearby smartphones by creating a small *ad hoc* smartphone cluster [15]. This would then allow the smartphone to perform a decoding as mentioned in this work and achieve higher transmission rates. The size of the cluster would be constantly changing, due to the *ad hoc* nature of the smartphones within range. As such the cluster would need to dynamically reassign load to suit the current architecture of the *ad hoc* cluster.

To communicate the decoding or encoding information between smartphones it is necessary to use some wireless mechanism. This obviously is a problem as you are using an unreliable transmission link to facilitate your encoding or decoding. The overheads to facilitate the wireless communication could severely diminish the performance benefit of the parallel LDPC product code decoder. The justification for this is that the smaller segments to be decoded by each smartphone need not under go as powerful a coding technique, due to the proximity of the smartphones to one another.Furthermore, it would be likely that such an application may be built at the application layer and all transmission between the smartphones would be built on reliable transfer protocols (such as Transmission Control Protocol (TCP) or Hyper Text Transfer Protocol (HTTP)). Therefore, the inter-smartphone communication becomes less of a concern and the focus falls on the more powerful LDPC product code. This application could allow small portable devices to perform processing tasks previously limited to the realm of dedicated servers and supercomputers.

6.6 Conclusion

In this dissertation a design of a parallel decoder for LDPC product codes is presented. The decoder is based on the belief-propagation algorithm by Judea Pearl [10]. The implemented decoder was shown to offer a 0.16 dB worsening on decoding BER performance for a 7.26 times decoding time performance increase. The decoder was deemed successful as it met its requirement of improving the decoding time performance whilst maintaining decent decoding BER performance.

Bibliography

[1] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*, chap. 3, pp. 245–246. Pearson, 5th ed., 2005.

[2] S. Lin, D. Costello, and M. Miller. "Automatic-repeat-request error-control schemes." *IEEE Communications Magazine*, vol. 22, no. 12, pp. 5–17, 1984.

[3] W. Huffman and V. Pless. Cambridge University Press, 2003.

[4] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Pearson-Prentice Hall, 2004.

[5] P. Elias. "Error-free coding." *IEEE Transactions on Information Theory*, vol. 4, no. 4, pp. 29–37, 1954.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1." In *IEEE International Conference on Communications*, vol. 2, pp. 1064–1070. IEEE, 1993.

[7] T. Moon. *Error Correction Coding*. John Wiley & Sons, 2005.

[8] D. Divsalar and E. Pollara. "Low-rate turbo codes for deep-space communications." In *IEEE International Symposium on Information Theory*, p. 35. IEEE, 1995.

[9] H. Dai and H. Poor. "Advanced signal processing for power line communications." *IEEE Communications Magazine*, vol. 41, no. 5, pp. 100–107, 2003.

[10] J. Pearl. "Reverend Bayes on inference engines: A distributed hierarchical approach." *Proceedings of the Second National Conference on Artificial Intelligence*, pp. 133–136, 1982.

[11] M. Fuechsle, J. Miwa, S. Mahapatra, H. Ryu, S. Lee, O. Warschkow, L. Hollenberg, G. Klimeck, and M. Simmons. "A single-atom transistor." *Nature Nanotechnology*, vol. 7, no. 4, pp. 242–246, 2012.

[12] R. Schaller. "Moore's law: past, present and future." *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[13] M. Lundstrom. "Moore's law forever?" *Science*, vol. 299, no. 5604, pp. 210–211, 2003.

[14] F. Alecu. "Performance analysis of parallel algorithms." *Journal of Applied Quantitative Methods*, vol. 129, 2007.

[15] F. Busching, S. Schildt, and L. Wolf. "DroidCluster: Towards Smartphone Cluster Computing–The Streets are Paved with Potential Computer Clusters." In *International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 114–117. IEEE, 2012.

[16] H. Burton and E. Weldon Jr. "Cyclic product codes." *IEEE Transactions on Information Theory*, vol. 11, no. 3, pp. 433–439, 1965.

[17] R. Pyndiah. "Near-optimum decoding of product codes: Block turbo codes." *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1003–1010, 1998.

[18] S. Reddy. "On decoding iterated codes." *IEEE Transactions on Information Theory*, vol. 16, no. 5, pp. 624–627, 1970.

[19] S. Reddy and J. Robinson. "Random error and burst correction by iterated codes." *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 182–185, 1972.

[20] J. Jiang and K. Narayanan. "Iterative soft decoding of Reed-Solomon codes." *IEEE Communications Letters*, vol. 8, no. 4, pp. 244–246, 2004.

[21] C. Jego and W. Gross. "Turbo decoding of product codes based on the modified adaptive belief propagation algorithm." In *IEEE International Symposium on Information Theory*, pp. 641–644. IEEE, 2007.

[22] D. Chase. "Class of algorithms for decoding block codes with channel measurement information." *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170–182, 1972.

[23] R. Pyndiah, A. Glavieux, and C. Berrou. "Method for detecting information bits processed by concatenated block codes.", Oct. 8 1996. US Patent 5,563,897.

[24] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer. "Separable MAP filters for the decoding of product and concatenated codes." In *IEEE International Conference on Communications*, vol. 3, pp. 1740–1745. IEEE, 1993.

[25] J. Hagenauer, E. Offer, and L. Papke. "Iterative decoding of binary block and convolutional codes." *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, 1996.

[26] A. Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[27] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. "Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)." *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[28] R. Tanner. "A recursive approach to low complexity codes." *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[29] J. C. Moreira and P. G. Farrell. *Essentials of Error-Control Coding*. John wiley & Sons Ltd, 2006.

[30] S. Wang, S. Cheng, and Q. Wu. "A parallel decoding algorithm of LDPC codes using CUDA." In *Asilomar Conference on Signals, Systems and Computers*, pp. 171–175. IEEE, 2008.

[31] U. Dasgupta and K. Narayanan. "Parallel decoding of turbo codes using soft output T-algorithms." In *Vehicular Technology Conference*, vol. 3, pp. 1204 –1210 vol.3. 2000.

[32] Z. Sun, L. Zhang, and Y. Tian. "SF-MAX-Log-MAP parallel decoding algorithm and its application study in LTE." In *Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC), 2011*, vol. 2, pp. 885 –888. july 2011.

[33] I. Kale, M. Taskaldiran, and R. Morling. "Increasing the speed of parallel decoding of turbo codes." In *Research in Microelectronics and Electronics, 2009. PRIME 2009. Ph.D.*, pp. 304 –307. july 2009.

[34] G. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities." In *Proceedings of the spring joint computer conference*, pp. 483–485. ACM, 1967.

[35] M. Hill and M. Marty. "Amdahl's law in the multicore era." *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[36] NVIDIA Corporation. "NVIDIA CUDA Programming Guide.", November 2011.

[37] NVIDIA Corporation. "NVIDIA GeForce GTX 680." White Paper, 2012.

[38] Intel. "Intelő Core i7-2600 Processor (8M Cache, 3.40 GHz)." `http://ark.intel.com/products/52213`. Last Accessed 22-03-2012.

[39] R. J. Wilson. *Introduction to Graph Theory*. Oliver and Boyd, 1972.

[40] W. Mayeda. *Graph Theory*. John Wiley and Sons, Inc, 1972.

[41] E. Henley and R. Williams. *Graph theory in modern engineering: computer aided design, control, optimization, reliability analysis*, vol. 98. Academic Press, 1973.

[42] J. Chen and M. Fossorier. "Near optimum universal belief propagation based decoding of low-density parity check codes." *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406–414, 2002.

[43] M. Fossorier, M. Mihaljevic, and H. Imai. "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation." *IEEE Transactions onCommunications*, vol. 47, no. 5, pp. 673–680, 1999.

[44] M. Davey and D. MacKay. "Low density parity check codes over GF (q)." In *Information Theory Workshop*, pp. 70–71. IEEE, 1998.

[45] M. Rosenstein, J. Collins, and C. De Luca. "A practical method for calculating largest Lyapunov exponents from small data sets." *Physica D: Nonlinear Phenomena*, vol. 65, no. 1, pp. 117–134, 1993.

[46] Mathworks. "Matlab - The Language of Technical Computing." `http://www.mathworks.com/products/matlab/`, 2011. Last Accessed 2011-07-20.

[47] Z. Qi and N. C. Sum. "LDPC product codes." In *The Ninth International Conference on Communications Systems*, pp. 481 –483. sept. 2004.

[48] S. Kullback and R. A. Leibler. "On information and sufficiency." *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.