# LDPC Code and Decoder Design for Gbps Systems.

Jeremy Thorpe

October 16, 2002

**Abstract**

Low-density parity check codes offer near Shannon limit performance on many channels with very low-complexity decoding. Current state-of-the-art decoders perform this decoding in serial, or with a limited number of operations in parallel. However, for extremely high throughput systems, an available strategy is to perform all operations in a single iteration in parallel over the entire LDPC code graph. This thesis proposal deals with the two primary problems impeding the implementation of fully parallel decoder implementation, namely wiring complexity and logic complexity.

## 1 Background

Forward error-correcting codes are used in applications ranging from computer memory to CD's to cell phones. Highly efficient coding systems, capable of reliably decoding information transmitted over noisy channels at rates close to the Shannon limit, have historically been used first by space agencies in deep space missions and are now proliferating into the consumer market in wireless networking.

One of the recent advances in error-correcting systems is the (re-)discovery of iterative decoding algorithms, in which each codeword symbol is estimated first from the channel, and subsequently from other symbols based on the code structure. Such iterative algorithms, often referred to as *turbo-decoding*, *message passing*, or *belief propagation* have achieved an astonishing improvement in the data rate achievable over a fixed channel with fixed error tolerance. One class of iteratively decodable codes, low-density parity check (LDPC) codes, shows performance exceptionally close to the Shannon limit

## 2 Proposed Research

The structure of LDPC codes is specified by a graph in which there are variable nodes, check nodes, and edges connecting check nodes to variable nodes. The

standard decoding algorithm, known as belief propagation (BP), is specified with respect to this graph. In BP, likelihood functions are recursively computed by each node in the graph, and a message containing this information is transmitted along each edge.

One of the great promises of this algorithm is that it can in principle be implemented by fully parallel hardware. In such a scheme, the graph would be laid out on in two-dimensions in VLSI technology. Each node in the graph would be instantiated by a hardware module able to carry out a simple computation, and each edge would be instantiated by a wire connecting the variable node to the check node. Since the performance of LDPC codes depends strongly on the size and randomness of the associated graph, there are certain problems that must be solved before this scheme is practical.

In a "completely" random graph (of a given *degree profile*) with a random layout has each edge spans approximately the dimension of the graph. Since the number of nodes grows the same as the number of edges, it can easily be seen that the *wiring complexity* grows faster than the *logic complexity* for graphs of increasing size. On the other hand, for graphs that are explicitly constructed to have very short edges, theoretical predictions and simulations of codes on random graphs will not in general apply, and performance will suffer in practice. However, there exist well known algorithms such as simulated-annealing and greedy graph-construction algorithms which are able to trade a small amount of performance for a huge reduction in wiring complexity *(see publications)*. In addition, bounds on some basic graph-theoretic quantities related to code performance, such as *diameter* and *girth,* can be derived in terms of wiring complexity. I propose to investigate the behavior of various code-construction algorithms, and compare this performance to analytical bounds to the extent possible.

A separate problem that must be solved for parallel decoders to become practical involves the messages and computations themselves. According to the BP equations, each node must be able to at least add and multiply real numbers. In computer simulations, this is normally accomplished by floating-point arithmetic. However, floating point numbers are many bytes long, and floating point arithmetic units take many thousands of transistors. By contrast, one of the very first message-passing algorithm, referred to as Gallager Algorithm A, uses only a single bit as a message and has logic that can be implemented in a very small number of gates. Of course, this algorithm requires a much better channel than the BP algorithm to achieve the same output error probability. Not surprisingly, there does exist a trade-off between complexity and performance. I propose to investigate this trade-off and characterize it to the extent possible.