Iterative Decoder Architectures

Engling Yeo, Borivoje Nikolić, and Venkat Anantharam, University of California, Berkeley

ABSTRACT

Implementation constraints on iterative decoders applying message-passing algorithms are investigated. Serial implementations similar to traditional microprocessor datapaths are compared against architectures with multiple processing elements that exploit the inherent parallelism in the decoding algorithm. Turbo codes and low-density parity check codes, in particular, are evaluated in terms of their suitability for VLSI implementation in addition to their bit error rate performance as a function of signal-to-noise ratio. It is necessary to consider efficient realizations of iterative decoders when area, power, and throughput of the decoding implementation are constrained by practical design issues of communications receivers.

INTRODUCTION

Error correction algorithms are frequently evaluated by their bit error rate (BER) vs. signal-to-noise ratio (SNR) performance. In practice, the implementations of these algorithms are constrained by the formats and throughput/latency requirements of specific communications standards. A practical implementation of a given algorithm in either hardware or software is ultimately evaluated by its cost (silicon area), power, speed, latency, flexibility, and scalability.

Investigation of iterative decoder implementations is particularly interesting. Theoretically, these decoders promise the required BERs at significantly reduced SNRs, very close to the Shannon limit. However, these studies are based on complex code constructions with little consideration of implementation. Practical implementation constraints will put the code construction and encoder/decoder implementation in a different perspective.

This article provides a view of the very large scale integration (VLSI) constraints that affect implementation of iterative decoders employing belief propagation as realized through message-passing algorithms. Such decoders rely on the cooperative interaction between several *soft* decoders over several iterations. This work examines the two main requirements in these decoders: message computation and exchange of messages between the soft decoders. Both are specified within a unified graphical framework consisting of an interconnected network of variable nodes and constraint nodes [1]. The desire is to emphasize the implications of code construction techniques on decoder implementation, beyond just BER performance. In particular, the requirements of the two major examples, turbo decoders and low-density parity check (LDPC) decoders, are examined. Similar considerations apply to related extensions of these codes, such as block turbo codes or tornado codes.

COMPUTATIONAL REQUIREMENTS

Using the graphical framework associated with iterative decoders, each node corresponds to a processing element that evaluates the marginal function of its input messages. Although the decoding of iterative codes is usually described as a sum-product algorithm on probability values, the computation is performed in the log-probability domain. Expressing the product of probabilities with the sum of log-probabilities or log-likelihood ratios (LLRs) is preferred in implementation because it replaces multipliers with adders. However, evaluating the sum in the log-probability domain requires a combination of exponential and logarithmic functions. In order to simplify the implementation, the computation can be approximated with the maximum value of the input operands, followed by an additive correction factor determined by a table lookup.

An example of the sum-product algorithm processed in log-probability domain is the add-compare-select (ACS) recursion in a maximum *a posteriori* (MAP) decoder (Fig. 1a). The add operations evaluate the logarithm of two product terms, while the compare and select operations approximate the logarithm of a sum of exponentials. This approximation (called the max-log-MAP) leads to an implementation loss of about 0.5 dB in a turbo decoder system. However, adding a correction factor to the output of

0163-6804/03/$17.00 © 2003 IEEE

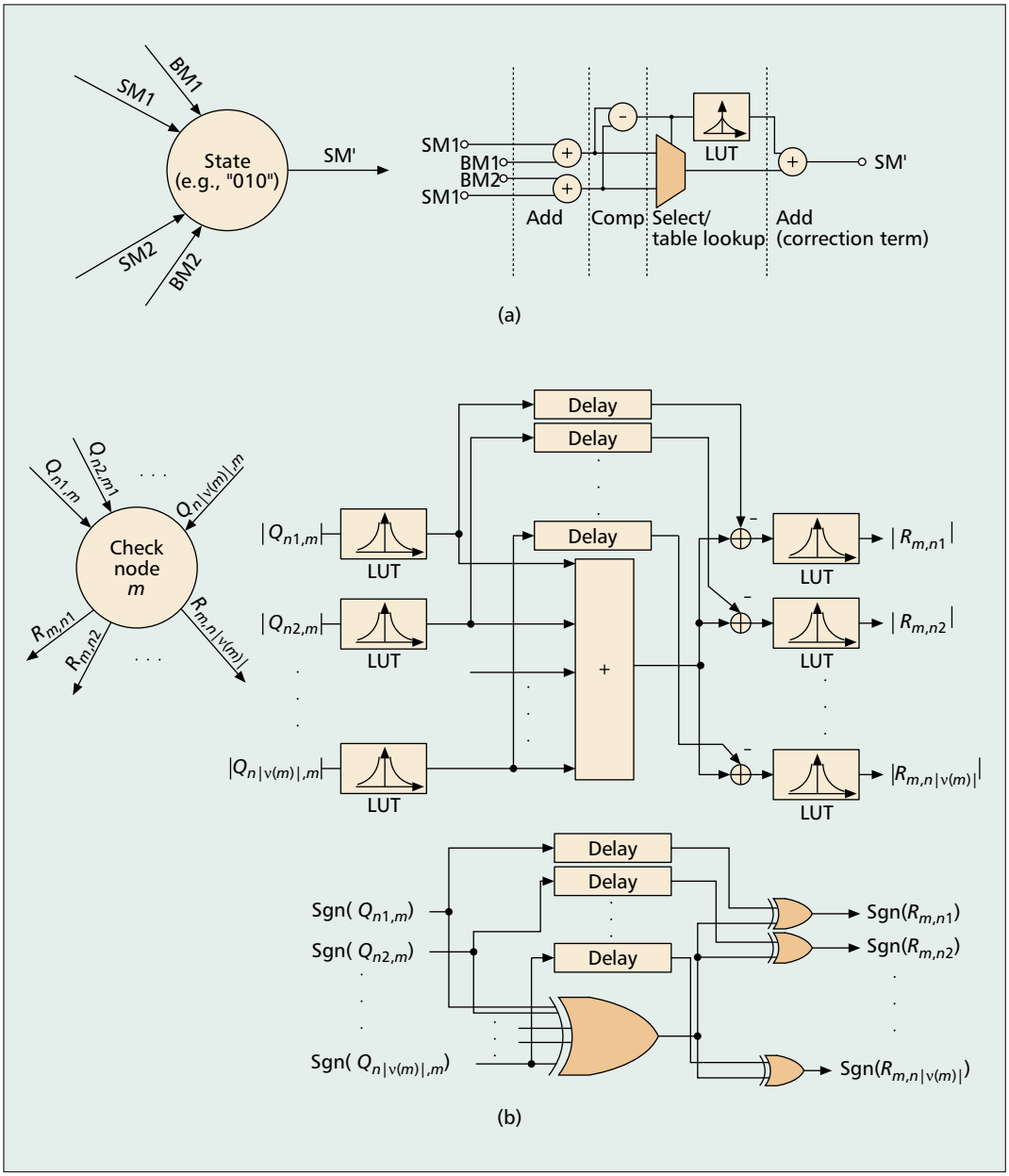

Figure 1. *Arithmetic computation associated with nodes in factor graphs of a) a convolutional code; b) an LDPC code.*

In order to simplify the implementation, the computation can be approximated with the maximum value of the input operands, followed by an additive correction factor, which is determined by a table lookup.

the ACS can restore the coding gain to within 0.1 dB of the MAP decoder performance [2]. This correction factor can be provided by a table lookup based on the difference of the two sums. Similar to commonly used Viterbi decoders, the throughputs of MAP decoders have been limited by implementation of the ACS structure due to single-step recursion, which prevents pipelining.

Another example of the sum-product computation in the log-probability domain can be found in LDPC decoders (Fig. 1b). The simple even parity check constraint evaluates

$$\prod_n (1-2p_n),$$

where p_n represents the probability that a bit $x_n = 1$. The use of log-probability domain simplifies the evaluation of the product, but also requires the table lookup evaluation of

$$\log\left[\tanh\left(\frac{LLR(x_n)}{2}\right)\right],$$

where $LLr(x_n)$ is defined as the log-likelihood ratio,

$$\log\left[\frac{p_n}{1-p_n}\right].$$

It has been shown that iterative decoders operating in the log-probability domain can frequently achieve good coding performance with arithmetic precision of just 3–5 bits. This implies that the lookup tables can be efficiently implemented with simple combinatorial logic functions directly implementing the required function.

A parallel realization of any algorithm will frequently be both throughput and power efficient, at the expense of increased area. On the other hand, serial realizations require fewer arithmetic units, and make use of memory elements in place of complex interconnect.

In addition to the calculations of marginal posterior functions, practical decoder implementations can lower the energy consumption per decoded bit by applying stopping criteria to the decoding iterations. This is done, for instance, in turbo codes by noting that the absolute log-likelihood values of all bits in a decoded block have exceeded a preset value, thus indicating sufficient confidence in the decoded output. Alternatively, the iterations in an LDPC decoder can be stopped when all parity check constraints have been met.

Message-Passing Requirements

The other key implementation feature of an iterative decoder is the interconnect network required to facilitate the exchange of messages between nodes in a factor graph. While each node in the graph is associated with a certain arithmetic computation, each edge in the graph defines the origin and destination of a particular message. The implementation of the message-passing requirements, however, appears in different forms for turbo and LDPC decoders.

The turbo decoder consists of a concatenation of MAP decoders separated by interleavers that permutate the sequence of inputs. Interleaving facilitates the exchange of messages between nodes that are adjacent in more than one of the underlying graphs. Although interleaving of messages can be performed through a direct-mapped network of interconnects for realization of a high-throughput interleaver, this will potentially result in intractable routing congestion due to the irregular nature of the interleaving network. In practice, the interleaving function is executed by writing the relayed messages sequentially into a random access memory array, and reading them out through a permuted sequence. The order of addresses used in the read access can be stored in separate read only memory (ROM) or computed on the fly. The latter method requires the addresses to be deterministically reproducible, and exploits the regularity in the interleaver structure to calculate addresses using, say, simple shifting or modulo division [3].

Likewise, an LDPC decoder is required to provide a network for messages to be passed between a large number of nodes. Direct wiring of the network leads to congestion in the interconnect fabric due to the disorganized nature of the defining graph. The congestion can be circumvented through the use of memory. Unlike the interleavers used in turbo codes, which have one-to-one connectivity, LDPC graphs have at least a few edges emanating from each variable node. The number of edges is several times larger than that in an interleaver network, and hence requires a larger memory bandwidth.

The practical implementation of a message-passing network is dependent on the structural properties of the graph. In general, the construction of good iterative codes requires large numbers of nodes whose interconnections are defined by graphs that are expanders and have a large girth [4]. These graphs tend to have a disorganized structure, which complicates the implementation of the message-passing network by requiring long global interconnects or memories accessed through an unstructured addressing pattern. More recently, graphs with structured patterns have emerged, [5], and they simplify the implementation of the decoders.

Iterative Architectures

The order of message computations and their exchange distinguishes two main classes of decoders: parallel and serial. Parallel decoder architectures directly map the nodes of a factor graph onto processing elements, and the edges of the graph onto an interconnect network. The parallel computation of messages requires the same number of processing elements as the number of nodes in the factor graph. On the other hand, serial decoder architectures distribute the arithmetic requirements sequentially among a small number of processing elements. Due to the sparseness of the graph, there is usually a delay between the generation and consumption of the messages. Hence, this technique requires additional memory elements to store the intermediate messages.

A parallel realization of any algorithm will frequently be both throughput- and power-efficient, at the expense of increased area. On the other hand, serial realizations require fewer arithmetic units, and make use of memory elements in place of complex interconnect.

Turbo Decoder Architectures

The MAP algorithm, used as a constituent of turbo decoders, was originally described as a serial recursion of forward/backward message passing. The most direct implementation of this serialized instance of belief propagation proceeds with forward propagation while storing the intermediate path metrics. When the end of the sequence is reached, the direction of recursion is reversed. Besides long decoding latency, this approach is also subject to a large memory requirement. For a convolutional code with constraint length K, the forward propagating metrics of the 2^{K-1} nodes at each bit instance in the trellis have to be stored until the corresponding backward propagating metrics are available for final processing. At a finer granularity, these 2^{K-1} ACS operations can either be performed by a group of parallel ACS processors as shown in Fig. 2 or serialized onto a single ACS processor. In serialized architectures, interconnect between separate ACS operators is substituted with memories or registers, but the overall throughput suffers accordingly.

A common implementation technique that achieves higher throughputs uses multiple windows [6]. This approach parallelizes the message passing algorithm over several subsets of bit sequences by partitioning the recursions into a number of independent overlapping windows. The shorter window lengths offer the advantage of lower latencies and reduced memory requirement for the intermediate path metrics. Since the propagating metrics do not cross into neighboring windows, the memory requirements are reduced to the amount required for temporary storage of propagating metrics with-

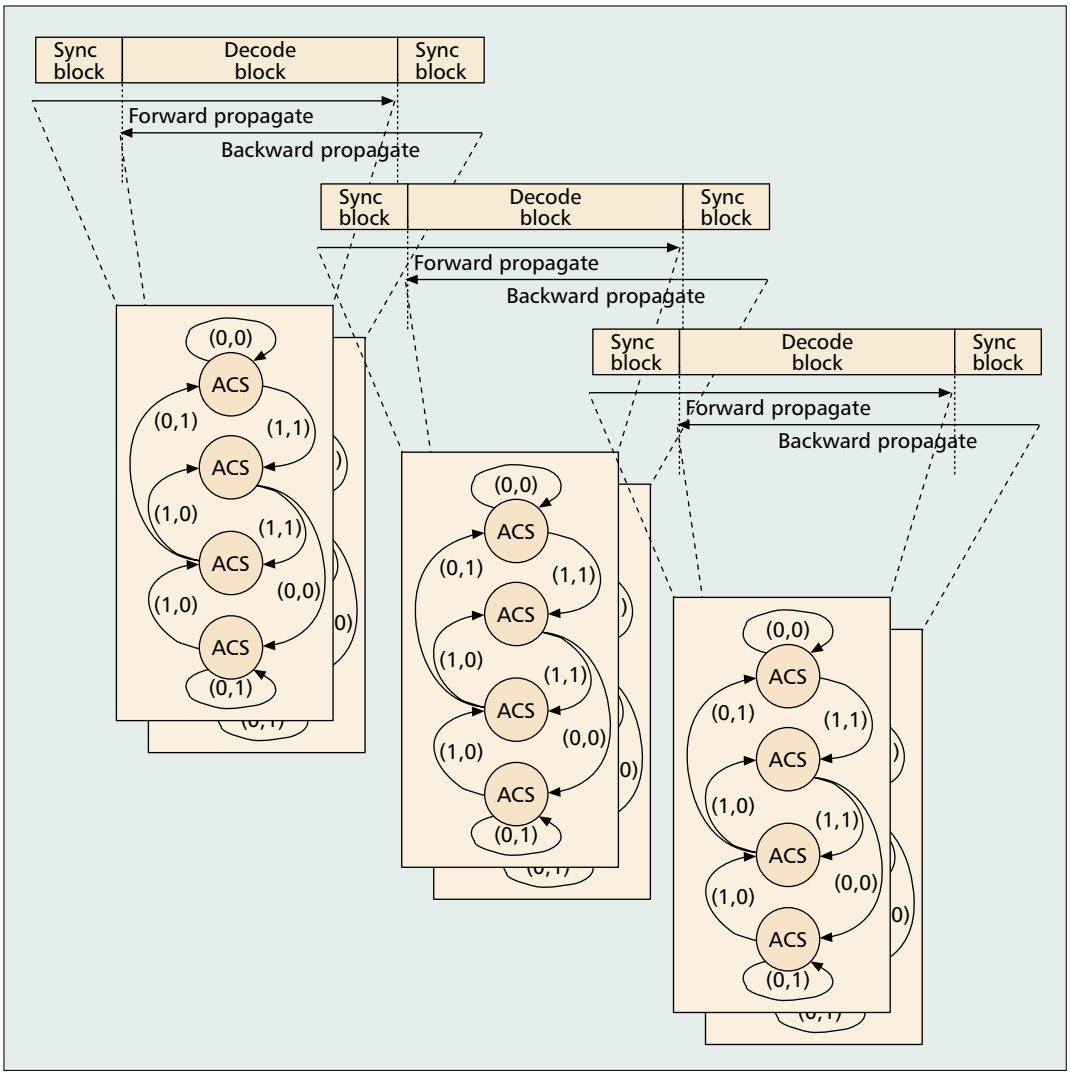

Figure 2. *Parallel architecture for windowed MAP.*

The overall throughput of a turbo decoder is also dependent on the implementation of the interleaver. Due to the size of the interleaver, the common approach uses memory blocks, but also places the memory access time in the critical path.

in one window. The overlapping regions are shown as sync blocks in Fig. 2, and their length, L, is usually set at about five times the constraint length of the convolutional code. The messages computed within the first L steps of each recursion window in either direction are typically discarded as they have insufficient accuracy. This is the arithmetic overhead incurred by each additional parallel window in a MAP decoder.

The overall throughput of a turbo decoder is also dependent on the implementation of the interleaver. Due to the size of the interleaver, the common approach uses memory blocks, but also places the memory access time in the critical path. Memory access is approximately 2 ns (general-purpose single-ported 32 kb memories in 0.13 µm complementary metal oxide semiconductor, CMOS, technology), significantly more than the 1 ns required to add two pairs of two short-wordlength numbers and select the maximum result in the ACS decoding logic (0.13 µm CMOS application-specific integrated circuit, ASIC, design). Decreasing average memory access time by increasing the number of I/O ports is unsuitable because it leads to quadratic growth in memory area. Hence, the memory access determines the serial symbol rate.

LDPC Decoder Architectures

In LDPC decoding, there is no interdependence between simultaneous variable-to-check and check-to-variable computations. Parallel LDPC decoders will benefit from throughput and power efficiency, but will require the implementation of a large number of processing elements together with message passing within a congested routing network. In order to ease the difficulty in routing, a common approach is to partition a design into smaller subsets with minimum overlap. However, due to irregularity in the parity check matrix, design partitioning is difficult and yields little advantage. An example of a parallel LDPC decoder [7] (Fig. 3a) for a 1024-bit rate-1/2 code requires 1536 processing elements with an excess of 26,000 interconnect wires to carry the messages between the processing elements. Area of implementation and interconnect routing are the two major issues inhibiting the implementation of parallel architectures.

In a serial LDPC decoder (Fig. 3b), the task of message computation is serialized into a small

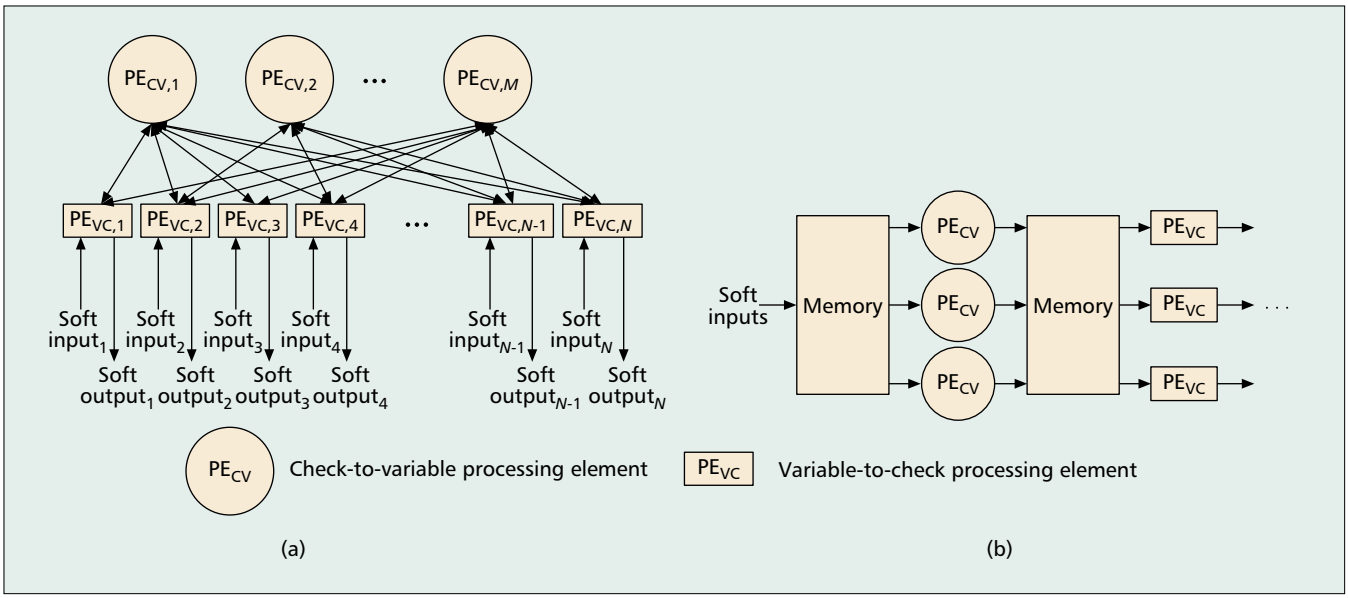

Figure 3. *a) Parallel vs. b) serial architectures for LDPC decoding.*

Standard	Application	Iterative code	Max. throughput
DVB-RCS	Digital video broadcast	Parallel conc. of 8-state conv. codes	68 Mb/s (rate 7/8)
IEEE 802.16	Wireless networking (MAN)	Turbo product codes	25 Mb/s (rate 5/6)
3GPP UMTS	Wireless cellular	Parallel conc. of 8-state conv. codes	2 Mb/s (rate 1/3)
CCSDS	Space telemetry	Parallel conc. of 16-state conv. codes	384 kb/s (rate 1/2)

Table 1. *Standard specifications for turbo decoding.*

number of processing elements, resulting in a latency of several thousand cycles. Furthermore, the large expansion property of LDPC codes with good asymptotic performance leads to stalls in processing between the rounds of iteration due to data dependencies. In order to capitalize on all hardware resources, it is more efficient to schedule all available processing elements to compute the messages in each round of decoding, storing the output messages temporarily in memory, before proceeding to a subsequent round of computations. Although serial architectures result in less area and routing congestion, they lead to dramatically increased memory requirements. The size of the memory required is dependent on the total number of edges in the particular code design, which is the product of the average edge degree per bit node and the number of bits in each block of LDPC code. For example, a serial implementation of a rate-8/9 4608-bit LDPC decoder with variable nodes with an average edge degree of four will have more than 18,000 edges in the underlying graph. It would have to perform 37,000 memory read or write operations for each iteration of decoding, which limits the total throughput. This is in contrast to a turbo decoder, whose memory requirement is largely dictated by the size of the interleaver required to store one block of messages. Given the same block size, the memory requirements for the serial implementation of an LDPC decoder are several times larger than that of a turbo decoder.

Platforms for Iterative Decoding

Table 1 lists some recent communication standards specifying the use of iterative decoding for forward error correction. Iterative codes are also currently being considered for high-performance storage applications that require about 1 Gb/s throughput as well as 10 Gb/s optical communications. Iterative decoding for most applications can be implemented on a number of platforms. The choice of platform is dictated primarily by performance constraints such as throughput, power, area, and latency, as well as two often understated and intangible considerations: flexibility and scalability. Flexibility of a platform represents the ease with which an implementation can be updated for changes in the target specification. Scalability captures the ease of using the same platform for extensions of the application that may require higher throughputs, increased code block sizes, higher edge degrees for low-density parity check codes, or increased number of states in the constituent convolutional code of the turbo system.

General-purpose microprocessors and digital signal processors (DSPs) have a limited number of single-instruction-per-cycle execution units but provide the most flexibility. These platforms naturally implement the serial architecture for iterative decoding. Microprocessors and DSPs are used as tools by the majority of researchers

in this field to design, simulate, and perform comparative analysis of iterative codes. Performing simulations with BERs below 10^{-6}, however, is a lengthy process on such platforms. Recently, there has been increased momentum in the use of DSPs in third-generation (3G) wireless devices. The specifications require turbo decoding at throughputs up to 2 Mb/s, which is an order of magnitude faster than rates typically achievable by a handful of execution units. Advanced DSPs include a *turbo coprocessor*, which is essentially an ASIC accelerator with limited programmability.

Field programmable gate arrays (FPGAs) offer more opportunities for parallelism with reduced flexibility. However, fully parallel decoders face mismatch between the routing requirements of the programmable interconnect fabric and edges in a factor graph. FPGAs are intended for datapath-intensive designs, and thus have an interconnect grid optimized for local routing. The disorganized nature of an LDPC or interleaver graph, for instance, requires global and significantly longer routing. Existing implementations of iterative decoders on FPGAs continue to circumvent this problem by using time-shared hardware and memories in place of interconnect. This serial method currently limits the internal throughput of turbo decoders to 6.5 Mb/s [8] and LDPC decoders to 56 Mb/s [9].

Custom ASICs are well suited for direct mapped architectures, offering even higher performance with further reduction in flexibility. An LDPC decoder [7] implemented in 0.16 µm CMOS technology achieves 1 Gb/s throughput by fully exploiting the parallelism in the LDPC decoding algorithm. The logic density of this implementation is limited to only 50 percent to accommodate a large on-chip interconnect. In addition, the parallel architecture is not easily scalable to codes with larger block sizes. For decoding within 0.1 dB of the capacity bound, block sizes with tens of thousands of bits are required [10]. With at least 10 times more interconnect wires, a parallel implementation will face imminent routing congestion, and may exceed viable chip areas.

Current ASIC implementations of turbo decoders [3] are serial, targeting wireless applications. Decoding throughput is 2 Mb/s with 10 iterations of the two constituent convolutional decoders. A high-throughput ASIC turbo decoder, limited by the interleaver memory access, should be able to decode at throughputs over 500 Mb/s.

An alternative form of custom ASIC platform is based on analog signal processing [11]. Initial analog implementations of MAP decoders have reduced silicon area and achieved high decoding throughput. Analog decoders operate in the probability domain, evaluating the sum-product algorithm in its raw form, as opposed to operating in the log-probability domain in which digital decoders operate. Probabilities, represented as currents in bipolar junction transistors, are multiplied using a six-transistor Gilbert cell that takes advantage of the exponential relationship between collector current and base emitter voltage. Currently, analog MAP decoders only exist in fully parallel architectures, with small block sizes up to a few hundred bits. For a serial architecture with larger block sizes, a track-and-hold circuit can be employed to implement a one-step recursion similar to the ACS recursion in digital implementations. Despite their benefits, analog implementations are sensitive to process and temperature variations, and difficult to test in production. Analog circuits are also less scalable with improvements in process technology. Figure 4 and Table 2 provide summaries of the computational platforms and their performance.

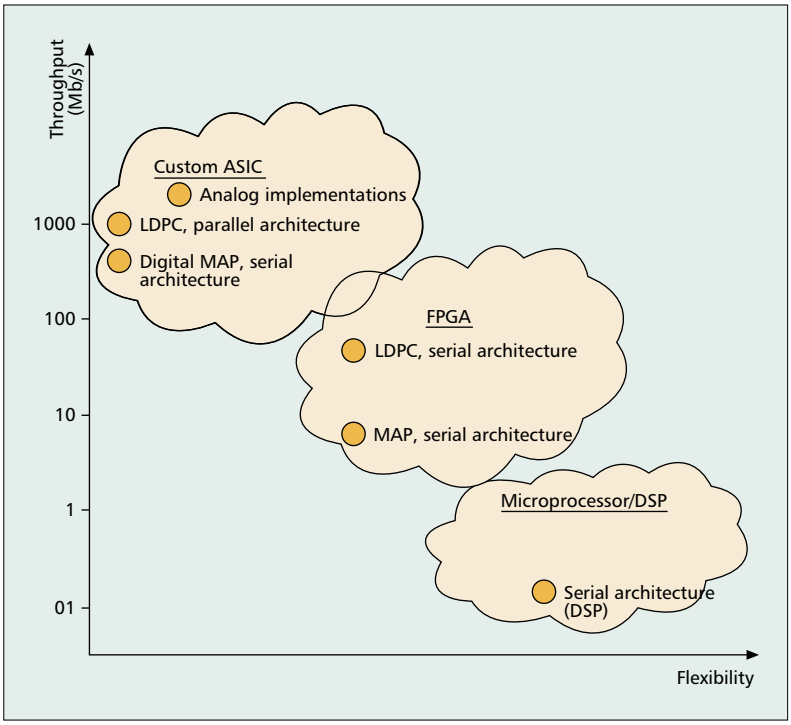

Figure 4. *Relative comparisons between various architectures on different platforms for implementation of iterative decoding.*

Impact of Code Construction on Decoder Architectures

The desire for large SNR gains frequently conflicts with the requirements for low complexity and high flexibility of the decoder. In most classes of iterative decoders, the properties that dominate the architectural considerations are size of the block code and number of iterations. In general, the BER performance of a code improves as the value of these numbers increase. However, given a block code with good expansion properties, decoding commences only after the final symbol in the block is received. A large block size not only imposes heavy computational and memory requirements on the decoder, but also leads to extended latencies. Likewise, a large number of iterations increases decoder latency and power while lowering effective throughput. In serial architectures, already slower due to the limited number of processing elements, each additional iteration requires more pipelined hardware in order to sustain the throughput rate. This results in further increase

Platform	Architecture	Example implementations	Implementation difficulty
Microprocessor/ DSP	Serial	133 kb/s rate-1/2 LDPC decoder on DSP [15]	Limited number of processing units (ALU)
FPGA	Parallel	None	Mismatch of interconnect requirements and capabilities
FPGA	Serial	56 Mb/s rate-1/2 LDPC decoder [9] 6.5 Mb/s 8-state MAP decoder [8] (3 windows)	Control for memory access
Custom ASIC	Parallel	1 Gb/s rate-1/2 LDPC decoder [7 1024-bit code block	Routing congestion; not scalable
Custom ASIC	Serial	2 Mb/s 8-state MAP decoder [3]	Interleaver addresses computed on the fly. Implementation was optimized for low power. 500 Mb/s high throughput MAP decoder is theoretically feasible
Custom ASIC (Analog)	Parallel	Analog MAP decoder in BiCMOS technology [11]	Interleavers not included. Sensitive to process and temperature variations. Difficult to test in production. Not scalable with improvements in process technology

Table 2. *A summary of platforms for iterative decoders.*

in area. Decoders that require above 1000 iterations or 10^7 processing elements [10] are not suitable for parallel architectures. Large codes are much more easily mapped onto serial architectures, but will result in extended decoding latencies.

Turbo Codes

The properties that are specific to the implementation of turbo decoders are the code constraint lengths of the constituent convolutional codes, as well as the design of the interleaver.

The constraint length K of a convolutional code directly affects the number of ACS operations required for either the forward or backward recursion in the MAP decoder. Turbo codes that comprise constituent codes with large constraint lengths have been shown to exhibit improved abilities to converge to the maximum likelihood solution. On the other hand, the constraint length, K, has an exponential effect on the area and power dissipation of the decoder, since the trellis representation has 2^{K-1} nodes at each bit instance. Most research in turbo codes has adopted constituent codes with moderate constraint lengths of less than five, and focused on the design of interleavers for performance improvements. Applications with low power or high throughput requirements will need to have low constraint lengths. As an example, the 3G Partnership Project (3GPP) Universal Mobile Telecommunications System (UMTS) specification uses convolutional codes with a constraint length of four.

As previously mentioned, the implementation of the interleaver becomes the throughput bottleneck when routing congestion prevents a direct-mapped network of interconnects and requires the use of memory. An example of this is the interleaver proposed in the original turbo code (see the article in this issue by C. Berrou), which has a size of 65536. The use of memory invokes a design decision over how the addresses of the memory accesses should be generated. Random interleavers require that the permutation addresses be stored in a ROM section. The size of the memory required for the permutation sequence is often larger than that required for storage of the messages. For instance, the interleaver with 65,536 messages would require 16 bits for the memory addresses, which is up to four times larger than the 4 or 5 bits typically used for fixed-point representation of the messages in turbo decoders.

A deterministic interleaver, on the other hand, presents possibilities of address generation on the fly, thus eliminating the need for ROM. For example, the permutation sequence can be linearly produced using just additions and modulo divisions, as shown in [12]. This simplification comes at some cost in BER performance. Compared with an S-random interleaver, a benchmark often used in interleaver designs, a degradation of 0.5 dB is reportedly measured at a BER of 10^{-5}. While more sophisticated deterministic interleavers exist, the cost of additional area and power usually offsets any benefits of a complex compute-on-the-fly strategy.

LDPC Codes

In terms of implementation, LDPC codes can be differentiated along the lines of whether the code has a structured graph, a uniform edge degree (regular codes), and the maximum edge degree of both check and variable nodes.

The structure of an LDPC graph, analogous to the structure of the interleaver in a turbo code, directly affects the implementation of the message-passing network. A graph with disorganized connections will lead to either routing congestion in parallel architectures or address indexing issues in serial architectures. However, there are examples of LDPC codes with highly structured and regular graphs. These codes are based on properties of finite fields [5] and exhibit a natural cyclic structure, which can be exploited to allow the use of fast and small shift registers. Column splitting on these codes also yields added parallelism between memory accesses in serial architectures with a

limited number of parallel processing elements [13]. The demonstrated rate-1/2 (64,32) code has a block size of 8190 bits, and achieves a BER of 10^{-5} at 1.8 dB away from the theoretical bound.

The edge degree of a code corresponds to the number of inputs or outputs on the processing elements. This property has similar effects as the constraint length of a constituent convolutional code in turbo systems because it determines the relationship between a variable node and its adjacent neighbors. Practical implementations of LDPC decoders, particularly parallel ones, benefit from regular code with a small maximum edge degree in order to avoid detrimental arithmetic precision effects and the complexity of collating a large number of inputs and outputs at the processing elements. LDPC codes generated from the method of density evolution [10] face these issues because the codes have maximum variable degrees on the order of a hundred. To avoid truncation of results, the output of a 100-input adder would require at least 7 bits more than the wordlength of its inputs, effectively doubling the wordlength of a 5-bit decoder implementation. In addition, a parallel architecture of the decoder would have to emanate at least 100 signal buses from some of the processing elements. Despite one of the best reported performances, at only 0.0045 dB away from the theoretical Shannon bound, such codes are not particularly suited to the realization of a parallel VLSI decoder.

In serial LDPC decoder architectures, the edge degree of the variable nodes determines the total number of edges in the graph, which affects the size of the memory required, as previously noted. A method for reduction of the memory requirement in LDPC decoders involves a combination of a staggered decoding schedule and an approximation of the variable-to-check message computation [13]. A small number of processing elements compute a serial stream of check-to-variable messages. Unlike traditional decoding methods, each variable-to-check message is approximated by the log-likelihood ratio of the variable, and marginalization of the variable-to-check messages is not performed. The log-likelihood ratio is updated (incremented) as soon as there is any available corresponding check-to-variable message. Compared to the classical message-passing algorithm, this method allows decoders with area or power constraints that limit the number of iterations to five or less to benefit from more than 75 percent reduction in memory requirement, at the expense of less than 0.5 dB loss in coding gain. The decoder has a memory requirement dependent only on the total number of variable nodes in the block. It is noted that the staggered decoding will not achieve the same asymptotic results as LDPC decoding under belief propagation.

In order to produce viable real-time iterative decoding, future decoders have to aggressively exploit the power and throughput advantages of parallel architectures. However, these have to be preceded by techniques in code construction that address the complexity of routing parallel implementations. Codes suitable for use in communications receivers may have to trade off the SNR performance for improved partitioning properties that would assist with the routing problem. An example in this direction makes use of simulated annealing to minimize the total length of interconnect while maximizing the total girth of the graph [14].

Conclusion

Implementations of iterative codes have to address the arithmetic requirements for a variety of sum-product algorithms and provide a network for message passing. While the computational requirements of iterative decoders have received attention in the research on iterative decoder implementations, the realization of the network to facilitate message passing has been less emphasized. In general, the underlying graphs describing interleavers used in turbo codes and the parity check pattern in LDPC codes have an unstructured nature, which leads to routing congestion if they are directly mapped into a network of interconnects. These issues are not only specific to turbo and LDPC codes, on which this article has primarily focused, but also extend to other iterative codes, such as block turbo codes and tornado codes.

Routing congestion can be addressed through serial architectures with limited parallelism and through the use of memory arrays to temporarily store messages. However, besides a large memory requirement, serial architectures also place the memory access in the critical path of the decoder system, thus degrading the overall throughput by as much as an order of magnitude. The choice between a serial or parallel implementation is tied to the trade-off between memory or interconnect complexity concerns.

Historically, considerations for decoder implementations have always been preceded by efforts in code construction with the sole purpose of improving coding gain. Methods based on density evolution or expander graphs have shown promising BER performance, but lead to intractable implementation issues. Future practical implementations of high-throughput iterative decoders will require code construction techniques that trade off the error correcting performance for reduced implementation complexity.

Unlike traditional decoding methods, each variable-to-check message is approximated by the log-likelihood ratio of the variable and marginalization of the variable-to-check messages is not performed.

References

[1] F. Kschischang, B. Frey, and H. A. Loeliger, "Factor Graphs and the Sum-product Algorithm," *IEEE Trans. Info. Theory*, vol.47, Feb 2001, pp. 498–519.

[2] P. Robertson, E. Villebrun, and P. Hoeher, "A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms Operating in the Log Domain," *Proc. IEEE ICC*, Seattle, WA, June 18–22, 1995, pp. 1009–13.

[3] M. Bickerstaff *et al.*, "A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18 mm CMOS," *IEEE J. Solid-State Circuits*, vol. 37 no. 11, Nov. 2002, pp. 1555–64.

[4] M. Sipser and D. A. Spielman, "Expander Codes," *IEEE Trans. Info. Theory*, vol.42, Nov. 1996, pp. 1710–22.

[5] Y. Kou, S Lin, and M. P. C. Fossorier, "Low-Density Parity-check Codes Based on Finite Geometries: A Rediscovery and New Results," *IEEE Trans. Info. Theory*, vol. 47, no. 7, Nov. 2001, pp. 2711–36.

[6] Z. Wang, Z. Chi, and K. K. Parhi, "Area-Efficient High Speed Decoding Schemes for Turbo/MAP Decoders," *Proc. IEEE ICASSP*, Salt Lake City, UT, May 2001, pp. 2633–36.

Future practical implementations of high-throughput iterative decoders will require code construction techniques that trade off the error correcting performance for reduced implementation complexity.

[7] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, Mar. 2002, pp. 404–12.

[8] J. Steensma and C. Dick, "FPGA Implementation of a 3GPP Turbo Codec," *Proc IEEE 35th Asilomar Conf. Sig., Sys. and Comp.*, Pacific Grove, CA, Nov. 4–7, 2001, pp. 61–65.

[9] T. Zhang and K. Parhi, "A 56Mb/s (3,6)-Regular FPGA LDPC Decoder," *Proc. IEEE SIPS 2002*, San Diego, CA, Oct. 16–18, 2002, pp. 127–32.

[10] S. Chung *et al.*, "On the Design of Low-density Parity-check Codes within 0.0045 dB of the Shannon Limit," *IEEE Comm. Lett.*, vol. 5, Feb. 2001, pp. 58–60.

[11] H. A. Loeliger *et al.*, "Probability Propagation and Decoding in Analog VLSI," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 837–43.

[12] O. Y. Takeshita and D. J. Costello, Jr., "New Deterministic Interleaver Designs for Turbo-Codes," *IEEE Trans. Info. Theory*, IT-46, Sept. 2000, pp. 1988–2000.

[13] E. Yeo *et al.*, "High Throughput Low-density Parity-check Architectures," *Proc. IEEE GLOBECOM*, San Antonio, TX, Nov. 25–29, 2001, pp. 3019–24.

[14] J. Thorpe, "Design of LDPC Graphs for Hardware Implementation," *Proc. IEEE ISIT*, Lausanne, Switzerland, June 30–July 5, 2002, p. 483.

[15] T. Bhatt, K. Narayanan, and N. Kehtarnavaz, "Fixed Point DSP Implementation of Low-Density Parity Check Codes," *Proc. IEEE DSP2000*, Hunt, TX, Oct. 15–18, 2000.

BIOGRAPHIES

ENGLING YEO [S'96] (yeo@eecs.berkeley.edu) received B.S. and M.S. degrees in electrical engineering and computer science from the University of California at Berkeley i(UC Berkeley) n 1994 and 1995, respectively. He was at the DSO National Laboratories, Singapore, between 1996 and 1999. He returned to UC Berkeley, and received a Ph.D. degree in electrical engineering and computer science in 2003. He has served as a course consultant for National Technological University, and is currently with STMicroelectronics. His research is focused on high-performance architectures for communications and embedded systems, with particular interests in iterative decoders.

BORIVOJE NIKOLIC [S'93–M'99] received Dipl.Ing. and M.Sc. degrees in electrical engineering from the University of Belgrade, Yugoslavia, in 1992 and 1994, respectively, and a Ph.D. degree from UC Davis in 1999. He was on the faculty of the University of Belgrade from 1992 to 1996. He spent two years with Silicon Systems, Inc., Texas Instruments Storage Products Group, San Jose, California, working on disk drive signal processing electronics. In 1999 he joined the Department of Electrical Engineering and Computer Sciences, UC Berkeley as an assistant professor. His research activities include high-speed and low-power digital integrated circuits and VLSI implementation of communications and signal processing algorithms. He received the NSF CAREER award in 2003, College of Engineering Best Doctoral Dissertation Prize, and Anil K. Jain Prize for the Best Doctoral Dissertation in Electrical and Computer Engineering at UC Davis in 1999, as well as the City of Belgrade Award for the Best Diploma Thesis in 1992.

VENKAT ANANTHARAM [F] is on the faculty of the EECS department at UC Berkeley. He is a recipient of the Philips India Medal and the President of India Gold Medal from IIT Madras, an NSF Presidential Young Investigator award from the U.S. National Science Foundation, and an IBM Faculty Development award. He is a co-recipient of the 1998 Prize Paper award of the IEEE Information Theory Society and a co-recipient of the 2000 Stephen O. Rice Prize Paper award of the IEEE Communications Theory Society. He is a past associate editor for *IEEE Transactions on Information Theory*, *Annals of Applied Probability*, and *Markov Processes and Related Fields*, and current associate editor for *Queuing Systems : Theory and Applications*, and *Foundations and Trends in Communications and Information Theory*. He is a co-founder and consulting chief scientist at Dune Networks. His research interests include problems arising in coding theory, communications, communication networks, game theory, information theory, probability theory and its applications, queuing networks, and stochastic control.