UNIFIED DECODER ARCHITECTURE FOR LDPC/TURBO CODES

Yang Sun and Joseph R. Cavallaro

Department of Electrical and Computer Engineering
Rice University, Houston, TX 77005
Email: {ysun, cavallar}@rice.edu

ABSTRACT

Low-density parity-check (LDPC) codes on par with convolutional turbo codes (CTC) are two of the most powerful error correction codes known to perform very close to the Shannon limit. However, their different code structures usually lead to different hardware implementations. In this paper, we propose a unified decoder architecture that is capable of decoding both LDPC and turbo codes with a limited hardware overhead. We employ maximum *a posteriori* (MAP) algorithm as a bridge between LDPC and turbo codes. We represent LDPC codes as parallel concatenated single parity check (PCSPC) codes and propose a group sub-trellis (GST) decoding algorithm for the efficient decoding of PCSPC codes. This algorithm achieves about 2X improvement in the convergence speed and is more numerically robust than the classical "tanh" algorithm. What is more interesting is that we can generalize a unified trellis decoding algorithm for LDPC and turbo codes based on their trellis structures. We propose a reconfigurable computation kernel for log-MAP decoding of LDPC and turbo codes at a cost of $\sim$15% hardware overhead. Small lookup tables (LUTs) with 9 entries of 2-bit data are designed to implement the log-MAP algorithm. Fixed point (6:2) simulation results show that there is negligible or nearly no performance loss by using this LUT approximation compared to the ideal case. The proposed architecture results in scalable and flexible datapath units enabling parallel decoding of LDPC/turbo codes.

Index Terms— LDPC codes, Turbo codes, Log-MAP algorithm, SISO decoder, VLSI decoder architecture

1. INTRODUCTION

Turbo codes, introduced in 1993 [1], and low-density parity-check (LDPC) codes, invented in 1963 [2] have recieved tremendous attention in the coding community. Due to their excellent error correction capability and near-capacity performance, LDPC and turbo codes have been accepted in many of the current and next generation wireless standards, such as WiMax, 3GPP LTE, UMTS, DVB-S2 and WCDMA.

The success of LDPC and turbo codes is mainly due to the efficient iterative decoding algorithm. Many efficient VLSI architectures for LDPC decoders have been investigated [3, 4, 5, 6], as well as for turbo decoders [7, 8, 9, 10]. However, to the best of our knowledge, a generic decoder that can support both types of codes is still lacking in the literature.

It is known that these two families of codes have similarities. For example, they can both be represented as codes on graphs which define the constraints satisfied by codewords. Both families of codes are decoded in an iterative way by using the sum-product algorithm or belief propagation algorithm. A few researchers have tried to connect these two codes by applying turbo-like decoding algorithm for LDPC codes. Mansour and Shanbhag [3] propose an efficient turbo message passing algorithm for architecture-aware LDPC codes. Hocevar [4] suggests a similar layered decoding algorithm which treats the parity check matrix as horizontal layers and passes the extrinsic messages between layers to improve the performance. Zhang and Fossorier [11] discuss a shuffled belief propagation algorithm to achieve a faster decoding speed. Lu and Moura [12] propose to partition the Tanner graph into several trees and apply a turbo-like decoding algorithm in each tree for faster convergence rate. Dai *et al.* [6] propose a similar turbo-sum-product hybrid decoding algorithm for quasi-cyclic (QC) LDPC codes by splitting the parity check matrix into two sub-matrices where the information is exchanged.

The main idea of all these works is to apply the divide-and-conquer strategy to the iterative decoding of LDPC codes. Instead of using the standard two phase message passing algorithm, they all try to apply the turbo principle in LDPC decoding. Our work was motivated by these results. We unify LDPC and turbo codes as parallel concatenated codes. We treat a LDPC code as a trellis constrained code in which there are M trellises defined by a $M \times N$ parity check matrix. We divide the factor graph of LDPC codes into loop-free sub factor graphs where a log-MAP algorithm is employed locally. A new algorithm called group sub-trellis (GST) algorithm for the decoding of LDPC codes to achieve near optimal performance with a reduced hardware complexity will be presented. Furthermore, we propose a unified trellis decoding flow for both LDPC and turbo codes which leads to a flexible decoder architecture for LDPC/turbo joint decoding.

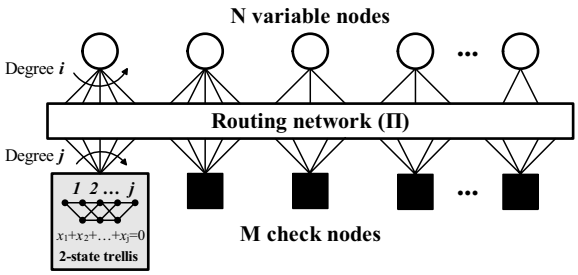

Fig. 1. Trellis structure for LDPC codes

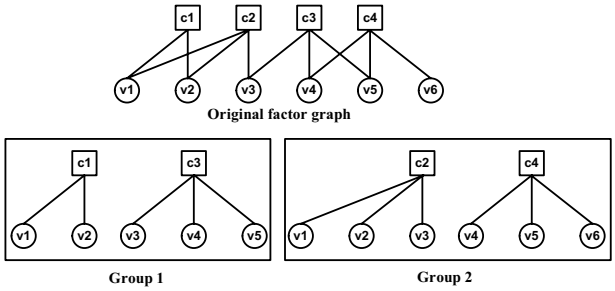

Fig. 2. Dividing a factor graph into groups

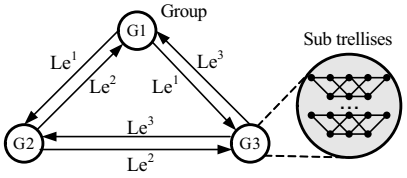

Fig. 3. GST messaging passing algorithm

2. ALGORITHM

2.1. Trellis structure for LDPC codes

The full trellis structure of an LDPC code is enormously large and it is impractical to apply the sum-product algorithm on it. However, alternately, a $M \times N$ LDPC code can be viewed as M parallel concatenated single parity check (PCSPC) codes. Fig. 1 illustrates a trellis representation for LDPC codes where a single parity check (SPC) code is considered as a low-weight 2-state trellis, starting at state 0 and ending at state 0. From this point of view, turbo codes are similar to LDPC codes in that turbo codes are two parallel concatenated N-state convolutional codes.

2.2. Group sub-trellis (GST) algorithm for LDPC codes

Generally, the performance of a single parity check code is poor. However, when many of them are sparsely connected they become a very strong code. Motivated by the turbo decoding principle, we divide the factor graph of an LDPC code into several groups which are loop-free. Each group is a subset of the graph (sub graph) which has simpler trellis structures so that the log-MAP algorithm can be applied on them. We refer to this algorithm as group sub-trellis (GST) decoding algorithm hereinafter. There are many ways to partition a factor graph, in this paper we only consider the **non-accessible** partition (meaning there are no paths between any two check nodes in each group as shown in Fig. 2). Fig. 3 shows an example of the extrinsic message passing between groups. In the GST algorithm, a faster convergence rate is expected because partial results generated by one group are used immediately by the next group. This is similar to turbo codes in the manner that each group is a constituent code and messages are passed iteratively between all the constituent codes.

2.3. Log-MAP algorithm for SPC codes

The main operation in the proposed GST algorithm is the log-MAP decoding of SPC trellis codes, where a SPC code could be represented as a terminated 2-state convolutional code as shown in Fig. 4. An efficient log-MAP decoding algorithm was given in [13]: for independent random variables x_0, x_1, ..., x_l the extrinsic message generated by a SPC code for x_i is

$$\Lambda(x_i) = L\Big(\sum_{\sim\{x_i\}} \oplus x_k\Big), \tag{1}$$

where the compact notation $\sim\{x_i\}$ represents the set of all the variables with x_i excluded. The function $L(\cdot)$ is associative and commutative, and is defined as

$$L(x_1 \oplus x_2) = \log \frac{1 + e^{L(x_1)} e^{L(x_2)}}{e^{L(x_1)} + e^{L(x_2)}}. \tag{2}$$

For simplicity, we use the notation $f(a, b)$ to represent the operation $L(x_1 \oplus x_2)$, where $a \triangleq L(x_1)$ and $b \triangleq L(x_2)$. Fig. 5 shows a forward/backward decoding flow and its corresponding decoder structure to implement (1). The forward (α) recursion and the backward (β) recursion are implemented as:

$$\alpha(i+1) = f(\alpha(i), \gamma(i)) \tag{3}$$

$$\beta(i) = f(\beta(i+1), \gamma(i+1)), \tag{4}$$

where $\gamma(i)$ is the branch metric and is equal to the sum of the channel LLR $L_{ch}(x_i)$ and the *a priori* information $L_a(x_i)$. The α and β states are initialized to $+\infty$ at the beginning, the extrinsic information for x_i is then computed as:

$$\Lambda(i) = f(\alpha(i), \beta(i)). \tag{5}$$

2.4. Log-MAP algorithm for turbo codes

We will not repeat the derivation of the log-MAP algorithm for N-state turbo codes, but just state the results. For more details, see [1]. Let S_k be the trellis state at time k, then the

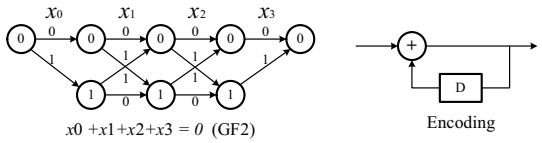

Fig. 4. Log-MAP decoding of SPC trellis codes

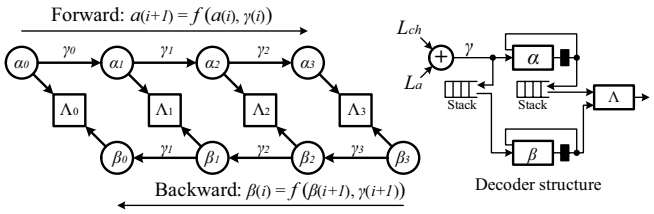

Fig. 5. Forward/backward recursion and its decoder structure

a posteriori probability (APP) of each information bit u_k is computed as:

$$L(\hat{u}_k) = \max_{\mathbf{u}:u_k=1}^{*} \{\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\} - \max_{\mathbf{u}:u_k=0}^{*} \{\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\}. \quad (6)$$

The forward-backward recursions are defined as:

$$\alpha_k(s_k) = \max_{s_{k-1}}^{*} \{\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k)\} \quad (7)$$

$$\beta_k(s_k) = \max_{s_{k+1}}^{*} \{\beta_{k+1}(s_{k+1}) + \gamma_k(s_k, s_{k+1})\}, \quad (8)$$

where γ_k is the transition probability and the $\max^*(\cdot)$ function is defined as

$$\max^{*}(a, b) = \max(a, b) + \log(1 + e^{-|a-b|}). \quad (9)$$

To implement (6-9) in hardware, a special add-compare-select (ACS) function unit is usually used [7].

3. ARCHITECTURE

3.1. Look-up-table approximation for LDPC decoding

To compute $f(a, b) = \log \frac{1+e^a e^b}{e^a+e^b}$ in hardware, we separate it into sign and magnitude calculation:

$$\text{sign}(f(a, b)) = \text{sign}(a)\,\text{sign}(b)$$
$$|f(a, b)| = f(|a|, |b|) = \min(|a|, |b|) + \log(1 + e^{-(|a|+|b|)}) - \log(1 + e^{-\left||a|-|b|\right|}).$$

Note that this $f(\cdot)$ function is mathematically equivalent to the classical "tanh" function $\Psi(x) = -\log(\tanh(|x/2|))$, but it is operating in a different domain. Due to its widely dynamic range (up to $+\infty$), the $\Psi(x)$ function has a high complexity and is prone to quantization noise. Though many approximations have been studied to improve the numerical accuracy of $\Psi(x)$ [14, 15, 16], it is still very expensive to implement it in hardware. However, the non-linear term in the $f(\cdot)$ function has a much lower dynamic range: $0 < g(x) \triangleq \log(1 + e^{-x}) < 0.7$, and thus is numerically more robust and less sensitive to quantization noise. It is interesting to know that $g(x)$ is exactly the same as the non-linear term in the turbo log-MAP algorithm (see (9)). To implement $g(x)$ in hardware, we propose to use a 4-value look-up-table (LUT) as shown in table 1. To translate this approximation into finite precision arithmetic, we propose to use a $q : 2$ quantization scheme (2 of q bits are used for the fractional part) which leads to a low complexity LUT as shown in table 2. In section 4.2 we will show this approximation leads to nearly no performance loss (< 0.05 dB) compared to the floating point data representation.

Table 1. Proposed LUT approximation

$\|x\|$	0	$0 < x \leq 0.75$	$0.75 < x \leq 2$	$x > 2$
$g(x)$	0.75	0.5	0.25	0

Table 2. Proposed 9-entry 2-bit LUT for $q : 2$ quantization

$\|x\|$	0	1	2	3	4	5	6	7	8	> 8
LUT	3	2	2	2	1	1	1	1	1	0

3.2. Reconfigurable kernel for LDPC/turbo decoding

Fig. 6(a) shows a logic circuit implementation for the LDPC $f(\cdot)$ function unit. As a comparison, the turbo ACS unit is also depicted in Fig. 6(b). Interestingly, they look very similar except for the position of the LUTs and the multiplexer. Fig. 6(c) shows the proposed architecture referred to as FACS (flexible ACS) for LDPC/turbo joint decoding. In terms of bit precision, the turbo ACS unit usually requires a higher bit precision for the internal α/β state metrics [17]. As a compromise, we conform the bit precision of $f(\cdot)$ to that of the turbo ACS unit. Also note that to improve timing, a "double-side" LUT (DLUT) is used to handle both positive and negative indices.

Assuming 10-bit precision is used for X and V and 8-bit precision is used for Y and W in the FACS unit, the synthesis results for a TSMC $90nm$ CMOS technology are summarized in table 3 in terms of maximum achievable frequency (assuming no clock skews) and area requirements at two frequencies. From table 3, about 15% area and timing overhead is observed for the proposed FACS unit.

3.3. Dual-mode SISO Engine

Based on the reconfigurable FACS unit, an LDPC and turbo dual-mode SISO decoder architecture is shown in Fig. 7. The

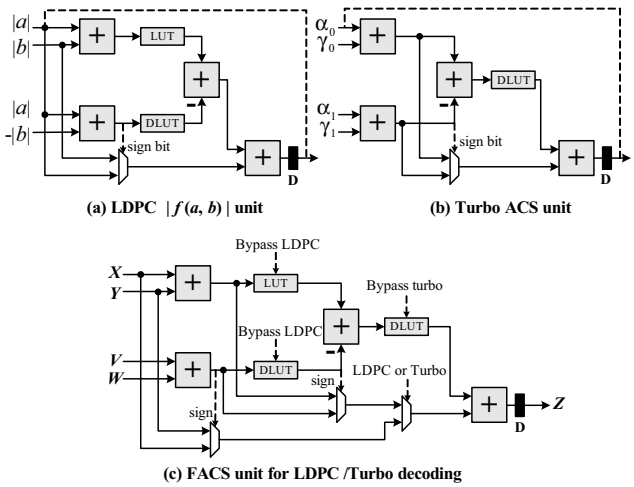

Fig. 6. Logic circuit implementation for LDPC/turbo codes

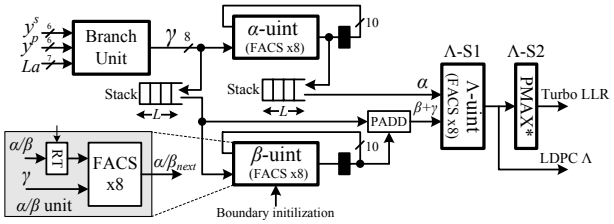

Fig. 7. SISO engine architecture (LDPC + turbo mode)

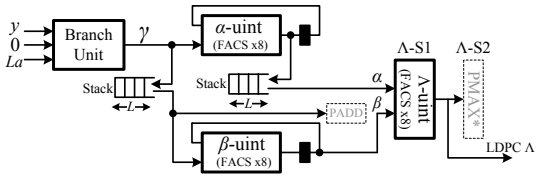

Fig. 8. SISO engine architecture (LDPC mode)

Table 3. Synthesis result for different kernels (90 nm)

Kernel	Max freq	Area@400MHz	Area@800MHz
FACS	820 MHz	1404 μm^2	2413 μm^2
ACS	890 MHz	1250 μm^2	2072 μm^2
$f(a,b)$	930 MHz	1182 μm^2	1876 μm^2

α/β recursion units, extrinsic Λ-$S1$ unit, branch unit, and memory stacks are shared between LDPC and turbo modes, saving substantial area. The SISO decoder is a reusable core and has been implemented in a 90 nm CMOS technology. The area distribution is summarized in table 4.

A. Turbo Mode: For an 8-state turbo decoder, all the elements of the architecture are used as shown in Fig. 7. We adopted a *Next Iteration Initialization* (NII) scheme, or a so called 1-α, 1-β scheme, as suggested in [18] and [19] in order to avoid the calculation of training sequences as initialization values for the β state metrics (the boundary metrics are initialized from the previous iteration). The α and β units implement (7) and (8) respectively based on two consecutive windows of data. Both α and β units compute 8 state metrics in parallel. A branch unit is used to compute the branch metrics γ based on the channel inputs (systematic y^s and parity y^p) and *a priori* information L_a. The α unit works in the natural order whereas the β unit and Λ unit work in the reverse order on the input data. Two stacks are used to delay and align data. To implement (6), we separate the LLR calculation into two cycles. Λ-$S1$ performs the first 8 ACS operations, and Λ-$S2$ performs 6 $\max^*(\cdot)$ and 1 subtraction operations.

B. LDPC Mode: For LDPC decoding, the decoder uses a subset of the architecture, as shown in Fig. 8 where the grayed blocks are not used. In the LDPC mode, the decoder can process 8 sub trellises simultaneously since each α, β and Λ-$S1$ block contains 8 FACS units.

A unified dataflow graph for LDPC/turbo decoding is shown in Fig. 9, in which the X-axis represents the trellis flow and the Y-axis represents the decoding time so that a box may represent the processing of a block of L data in L time steps. In turbo mode, L is equal to the sliding window size. In LDPC mode, L is equal to the length of one SPC trellis. The architecture can be reconfigured to support the decoding of 1) 8-state turbo codes, or 2) 8 SPC codes. During the decoding operation, the α-unit (in order), the β-unit (in reverse) and the Λ-unit (in reverse) work in parallel to achieve a real time decoding with a latency of L.

Table 4. SISO decoder area distribution @ 90 nm, 500 MHz

Unit	α	β	Λ-$S1$	Λ-$S2$	Stacks
Area (mm^2)	0.013	0.013	0.015	0.008	0.045

3.4. Top level architecture

For high throughput applications, multiple SISO decoders can be used in parallel to reduce the decoding latency and increase the decoding speed. Since parallel turbo decoder architectures have been well studied [8, 20, 9], in this section, we will focus more on the parallel implementation of the proposed GST algorithm for LDPC codes.

In the GST algorithm (see Fig. 3), suppose a factor graph is partitioned evenly into S groups where each group contains T number of 2-state sub trellises. Since each SISO decoder can process 8 sub trellises in parallel, we can dedicate $P = \lceil T/8 \rceil$ SISO decoders to process one group of sub trellises in parallel. As discussed before, message passing between groups is done iteratively to achieve a faster conver-

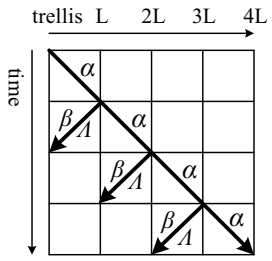

Fig. 9. Unified dataflow for LDPC/turbo joint decoding

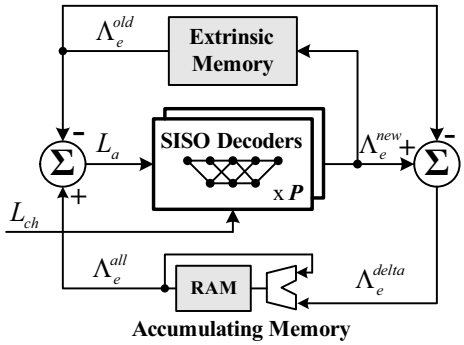

Fig. 10. Proposed GST decoder architecture

gence speed. Fig. 10 shows the proposed GST decoder architecture. In this architecture, one iteration consists of S sub iterations. Among these S sub iterations, P number of SISO decoders are used in a time-shared manner. Each group has an extrinsic memory to store the most recent extrinsic information generated by this group. During each sub-iteration, the *a priori* information L_a is formed by subtracting the old extrinsic information generated by this group from the total extrinsic information generate by all S groups:

$$L_a(u_k) = \Lambda_e^{all}(u_k) - \Lambda_e^{old}(u_k). \quad (10)$$

The new extrinsic information generated by this group based on (1) is then stored back to the extrinsic memory. An accumulating memory is used to save the column sum generated by S groups. The updating of the column sum is as follows:

$$\Lambda_e^{delta}(u_k) = \Lambda_e^{new}(u_k) - \Lambda_e^{old}(u_k) \quad (11)$$
$$\Lambda_e^{all}(u_k) = \Lambda_e^{all}(u_k) + \Lambda_e^{delta}(u_k). \quad (12)$$

The total memory requirement (in bits) for the extrinsic memory is equal to the total number of non-zero elements in the parity check matrix multiplied by the word length of Λ_e. And the total memory requirement for the accumulating memory is equal to the LDPC code size multiplied by the word length of Λ_e^{all}.

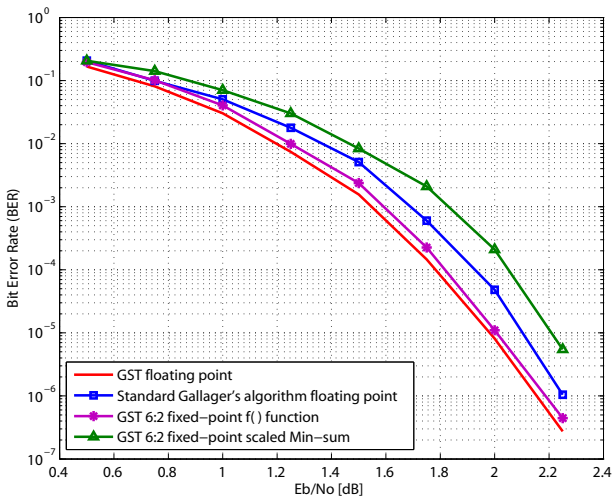

Fig. 11. BER performance comparisons (BPSK, AWGN)

4. RESULTS

4.1. ASIC synthesis result

Table 5 shows the ASIC synthesis results for some practical LDPC and turbo codes. In table 5, only the area of the SISO decoders and memories are shown (turbo interleaver and the LDPC permuter are not included).

4.2. Simulation result for LDPC codes

We show the simulation result for one WiMax LDPC code with code size = 2304 and code rate = 1/2. We divide its factor graph into 12 groups. Each group corresponds to one block row of the parity check matrix and contains 96 sub trellises. Fig. 11 compares the bit error rate (BER) performance of the GST decoder based on floating point and fixed point simulation with a maximum iteration $I = 15$. Also shown in the figure is the floating point BER of the standard Gallager's two-phase decoding algorithm. As can be seen from the figure, the proposed GST algorithm achieves better BER performance than the standard Gallager's two-phase algorithm when the maximum iteration is the same. This is because the GST algorithm has a faster convergence rate as shown in Fig. 12. The fixed point simulation result shows only a degradation of $<$ 0.05dB at BER 1x10^{-6} compared to the ideal performance, while a scaled (s=0.75) min-sum approximation has about 0.3dB degradation.

5. CONCLUSION

A unified decoder architecture for LDPC and turbo codes has been presented. Multi-mode decoding is achieved by employing a flexible FACS unit. By representing LDPC codes as parallel concatenated single parity check (PCSPC) codes, we

Table 5. Synthesis results for some practical LDPC and turbo codes @ $90nm$ technology, 500 MHz

Code type	Code size	Parallelism	SISOs	Quant.	Max iter.	SISO area	Memory area	Throughput
LDPC 802.16e	576 - 2304 bit	24 - 96	12	6:2	12	1.2 mm^2	1.06 mm^2	192 - 750 Mbps
LDPC 802.11n	648 - 1944 bit	27 - 81	11	6:2	12	1.1 mm^2	0.95 mm^2	180 - 640 Mbps
Turbo 3GPP-LTE	40 - 6144 bit	1 - 8	8	6:2	6	0.8 mm^2	1.2 mm^2	15 - 250 Mbps

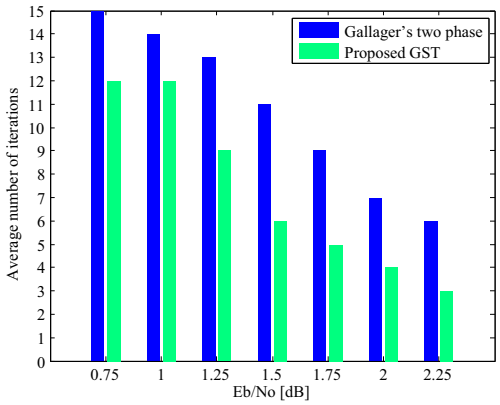

Fig. 12. Comparison of the convergence rate

propose a group sub-trellis (GST) decoding algorithm which achieves 2X improvement in the convergence speed. The unified LDPC/turbo architecture is attractive in supporting multi-standard communication systems.

6. ACKNOWLEDGEMENT

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

7. REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *IEEE Int. Conf. Commun.*, May 1993, pp. 1064–1070.

[2] R.G. Gallager, *Low-Density Parity-Check Codes*, MIT press edition, 1963.

[3] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Tran. VLSI Syst.*, vol. 11, pp. 976–996, Dec. 2003.

[4] D.E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *IEEE Workshop on Signal Processing Systems (SIPS)*, Oct 2004, pp. 107–112.

[5] Z. Wang and Q. Jia, "Low complexity, high speed decoder architecture for quasi-cyclic LDPC codes," in *IEEE Int. Symp. on Circuits and Systems*, May 2005, vol. 6, pp. 5786–5789.

[6] Y. Dai, Z. Yan, and N. Chen, "High-throughput turbo-sum-product decoding of QC LDPC codes," in *40th Annual Conf. on Info. Sciences and Syst.*, March . 2006, vol. 11, pp. 839–8446.

[7] G. Masera, G. Piccinini, M. Roch, and M. Zamboni, "VLSI architecture for turbo codes," in *IEEE Trans. VLSI Syst.*, 1999, vol. 7, pp. 369–3797.

[8] B. Bougard, A. Giulietti, L. Van der Perre, and F. Catthoor, "A class of power efficient VLSI architectures for high speed turbo-decoding," in *IEEE Conf. Global Telecommunications*, 2002, vol. 1, pp. 549 – 553.

[9] S-J. Lee, N.R. Shanbhag, and A.C. Singer, "Area-efficient high-throughput MAP decoder architectures," *IEEE Trans. VLSI Syst.*, vol. vol.13, pp. pp. 921–933, Aug. 2005.

[10] Y. Lin, S. Mahlke, T. Mudge, and C. Chakrabarti, "Design and Implementation of Turbo Decoders for Software Defined Radio," in *IEEE SIPS*, Oct. 2006, pp. 22–27.

[11] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," in *Asilomar Conf. on Signals, Systems and Computers*, Nov 2002, vol. 1, pp. 8–15.

[12] J. Lu and J.M.F. Moura, "Turbo like decoding of LDPC codes," in *IEEE Int. Conf. on Magnetics*, March 2003, pp. DT–11.

[13] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Tran. Info. Theory*, vol. 42, pp. 429–445, 1996.

[14] G. Masera, F. Quaglio, and F. Vacca, "Finite precision implementation of LDPC decoders," in *IEEE Proc. Commun.*, Dec 2005, vol. 152, pp. 1098– 1102.

[15] T. Zhang, Z. Wang, and K.K Parhi, "On finite precision implementation of low density parity check codes decoder," in *IEEE Int. Symposium on Circuits and Systems*, May 2001, vol. 4, pp. 202–205.

[16] D. Oh and K.K. Parhi, "Low complexity implementations of sum-product algorithm for decoding low-density parity-check codes," in *IEEE Workshop on Signal Processing Systems (SIPS)*, Oct 2006, pp. 262–267.

[17] Y. Wu, B. D. Woerner, and T. K. Blankenship, "Data Width Requirements in SISO Decoding With Modulo Normalization," in *IEEE Tran. Commun.*, 2001, vol. 49, pp. 1861–1868.

[18] J.Dielissen and J.Huisken, "State vector reduction for initialization of sliding windows MAP," in *2nd International Symposium on Turbo Codes and Related Topics*, Sept. 2000.

[19] A. Abbasfar and K. Yao, "An efficient and practical architecture for high speed turbo decoders," in *IEEE Vehicular Technology Conference*, 2003, vol. 1, pp. 337 – 341.

[20] G. Prescher, T. Gemmeke, and T.G. Noll, "A parametrizable low-power high-throughput turbo-decoder," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Mar. 2005, vol. 5, pp. 25–28.