AREA-EFFICIENT HIGH SPEED DECODING SCHEMES FOR TURBO/MAP DECODERS

Zhongfeng Wang[1], *Zhipei Chi*[2] *and Keshab K. Parhi*[2]

[1]MorphICs Technology Inc.
1550 South Bascom Avenue, Suite 200, Campbell, CA 95008
email: zwang@morphics.com

[2]Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, USA
email: {zchi, parhi}@ece.umn.edu

ABSTRACT

Turbo decoders inherently have a large latency and low throughput due to iterative decoding. To increase the throughput and reduce the latency, high speed decoding schemes have to be employed. In this paper, following a discussion on basic parallel decoding architectures, two types of area-efficient parallel decoding schemes are proposed. Detailed comparison on storage requirement, number of computation units and the overall decoding latency is provided for various decoding schemes with different levels of parallelism. Hybrid parallel decoding schemes are proposed as an attractive solution for very high level parallelism implementations. Simulation results demonstrate that the proposed area-efficient parallel decoding schemes introduce no performance degradation in general. The application of the pipeline-interleaving technique to parallel Turbo decoding architectures is presented at the end.

1. INTRODUCTION

Turbo codes invented by Berrou *et al* in 1993 [1] has been recognized as a breakthrough in the coding society. While achieving excellent decoding performance, turbo decoder suffers from a low decoding throughput inherently imposed by the iterative decoding feature.

A turbo decoder consists of two soft-input soft-output (SISO) decoders and an interleaver/ de-interleaver between them. Each constituent decoder has two outputs [1] at each time instant: (1) extrinsic information, denoted as $L_{ex}^{i}(k)$ where k represents the time index, $i = 1$ or 2 corresponds to ith decoder, (2) log likelihood ratio (LLR), denoted as $L_{lr}^{i}(k)$. The extrinsic information output from one constituent decoder is used as *a priori* information for the other constituent decoder after interleaving /de-interleaving.

To achieve multiple speed-up of throughput, parallel turbo decoding schemes have to be employed. Basically, two different classes of parallel schemes are available. Shown in Fig. 1 (a) and (b) are examples of two classes of parallel decoding schemes with twice speed-up. In general, if F-level parallelism is to be designed and the Class A parallel scheme is adopted, $F + 1$ sets of input buffers for $F + 1$ whole frames of data, F sets of SISO decoders and interleavers/de-interleavers are required. While with the Class B decoding scheme, only 2 sets of input buffers (for 2 data frames), F sets of SISO decoders and 1 set of interleaver /de-interleaver are required. Although the memory design for the Class B scheme is more complicated than Class A cases due to multiple address accesses in one clock cycle, the overall hardware complexity of Class B schemes is obviously much less than that of the Class A when $F > 2$. So, we will focus on the Class B parallel decoding scheme in this work.

This research was supported by Defense Advanced Research Project Agency under contract number DA/DABT63-96-C-0050.

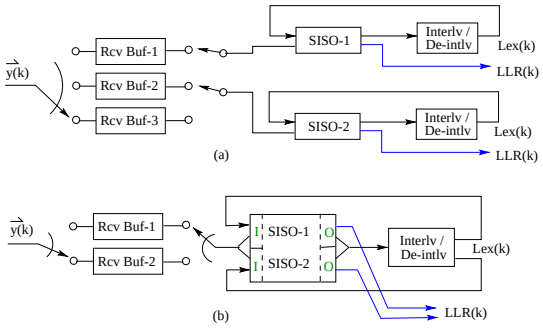

Figure 1. Block diagrams of two classes of parallel decoding schemes

This paper is organized as follows. In section 2, after presenting a modified version of existing parallel decoding scheme, two types of area-efficient parallel decoding schemes are proposed with detailed comparison on the storage requirement, number of computation units and the overall decoding latency. We argue that hybrid parallel decoding schemes seem to be the best choices for very high level parallelism implementations. In section 3, the application of the pipeline-interleaving technique to parallel turbo decoding architectures are addressed and simulation results are presented to show that the proposed area-efficient parallel decoding schemes have no performance degradation in general in section 4. The conclusion comes in the last section.

2. AREA-EFFICIENT PARALLEL DECODING SCHEMES

2.1. Segmented sliding window approach

A couple of parallel decoding schemes found in the literature was proposed in [2][3]. The basic idea is to divide a decoding frame into F sub-blocks. Global recursion operations for forward and backward state metrics are simultaneously performed on each sub-block. To ensure that the initial values of these recursion operations at intermediate points of a frame to be reliable, an overlap has to be considered between adjacent sub-blocks. An example of the parallel decoding scheme is shown in the top part of Fig. 2.

To reduce memory size for the storage of state metrics and also to reduce the latency, the sliding window approach [4] can be used for each segment of a decoding block (shown at the bottom part of Fig. 2). The cost is one more β computation unit for each segment of a block.

If F-level parallelism is assumed, totally F copies of α units, $2F$ copies of β units and F copies of LLR/L_{ex} computation units are required. The required memory for storage of computed for-

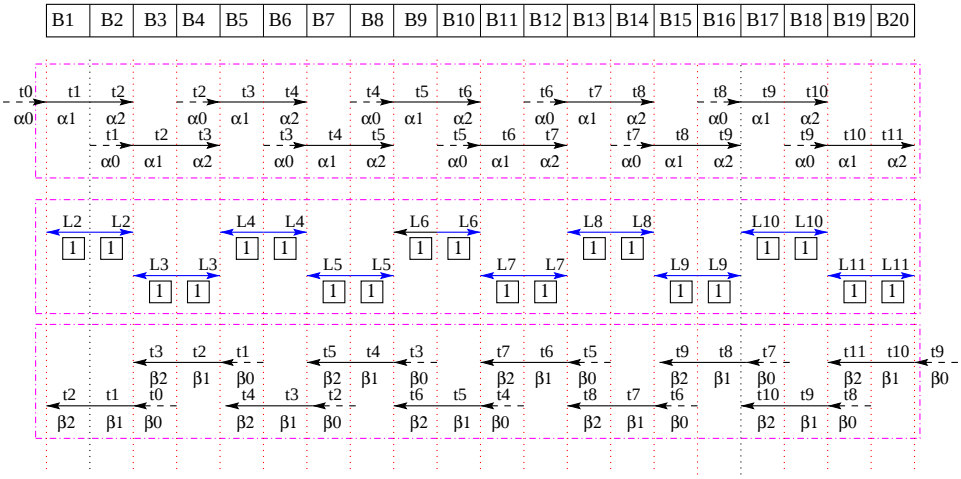

Figure 3. An example of an type-I 2-parallel turbo decoding scheme.

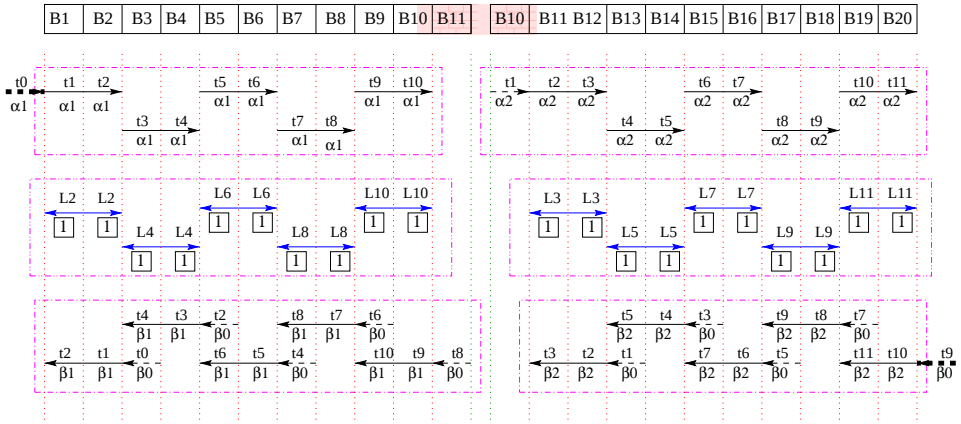

Figure 4. An example of a type-II 2-parallel turbo decoding scheme.

ward state metrics would be $L * F * M$ bits where L is the length of the sliding window and M is total number of state of one trellis stage. Assuming $M = 4$ (i.e., $K = 3$) and $F = 4$, the storage of state metrics would require $16 * 4 * 4 * 9 = 2304$ bits [5, 6], which is acceptable. The overall decoding cycles for one iterative decoding would be B_s+2 in general, where $B_s = N/F$ denotes the sub-block size.

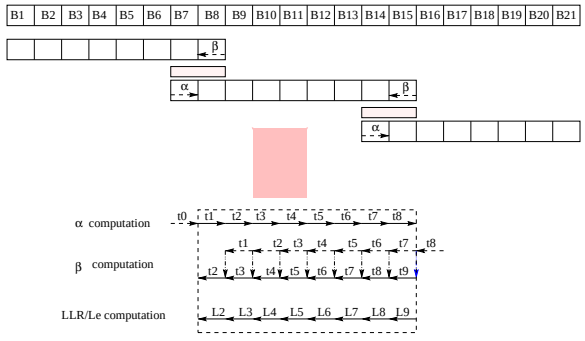

Figure 2. A conventional parallel decoding scheme and its modification

2.2. Area efficient parallel decoding schemes

The modified parallel decoding scheme proposed in section 2.1 (denoted as *segmented sliding window (SSW) approach* for convenience of later discussion) has a drawback that the ratio of pre-computation part to real computation of backward state metrics is 1:1. To achieve a high area efficiency, this ratio has to be reduced. In other words, once reliable backward or forward recursion state metrics have been obtained with a pre-computation unit working for a considerable amount of decoding cycles, the effective recursion operations starting at this point should be *continued for more decoding cycles.*

On the other hand, the recursive computation of forward recursion state metrics and backward recursion state metrics are symmetric. Therefore, we can either continuously compute forward recursion state metrics while computing backward recursion state metrics (sub) block by block using the sliding window approach, or we can continuously compute backward recursion state metrics while computing forward recursion state metrics (sub) block by block with the sliding window approach. Furthermore, we can also use the sliding window approach for both forward and backward recursion operations and this makes parallel decoding schemes more flexible. Therefore, two types of parallel decoding schemes can be constructed where we follow the rules below:

1. Decoding latency should should never be larger than $S + 2$ where S is total number sub-blocks;
2. The number of metrics/extrinsic information computation units doesn't exceed F at any time interval where F is the level of parallelism;
3. The number of memory needed doesn't exceed one segment at any time.

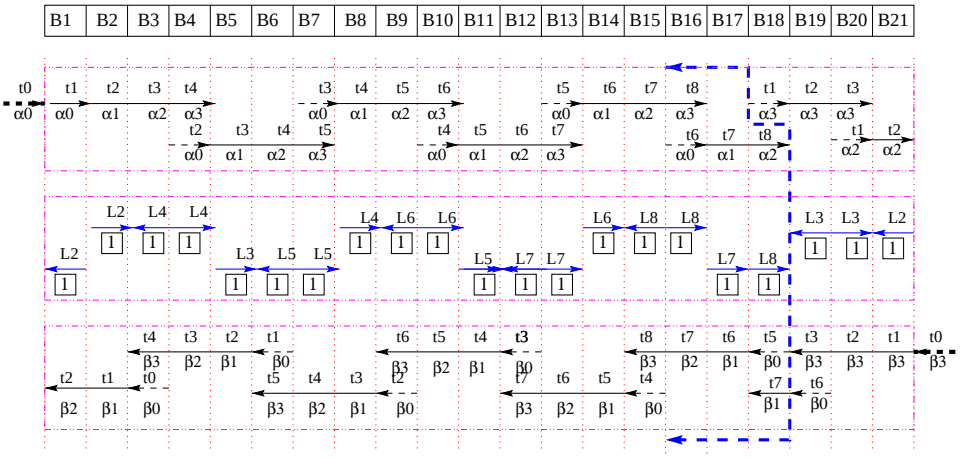

Figure 5. An example of a type-I 3-parallel turbo decoding scheme.

Shown in Fig. 3 is an example of a 2-parallel type-I parallel decoding scheme. All computation units work in a pipelined way over the entire frame. In the figure, the left-oriented arrows indicate forward recursive computation and the right-oriented arrows represent backward recursion. The dashed lines denote the pre-computation parts and solid lines represent real (effective) computation parts. Lt denotes performing the computation of LLR and extrinsic information during the time segment with index t. The small boxes underneath the oriented arrows represent the memory requirement for the storage of state metrics, *e.g.*, $\boxed{1}$ means that state metrics within one sliding window must be saved for one time segment and memory must be provided to save the state metrics with length of 1 sliding window. A time segment is referred to the time for the decoder to finish its decoding (within half iteration) of a sliding window.

It is not hard to find that the overall decoding cycles and computation units for this scheme are exactly the same as with the SSW approach. However, this parallel decoding scheme has a small benefit that it does not require the received data over a frame are available at the beginning of iterative decoding.

Shown in Fig. 4 is an example of a type-II parallel decoding scheme for 2-parallel decoding where a frame is divided into two parts, each part consists of the same number of sub-blocks including 1 shaded sub-block, which is the overlap part. All computation units work in a pipelined way over a sub-block instead of an entire frame.

It can be seen that, for this 2-parallel decoding scheme, two α units ($\alpha 1$ and $\alpha 2$ in the diagram) and three β units ($\beta 0$, $\beta 1$ and $\beta 2$) and two LLR/L_{ex} computation units are required.

Compared with the SSW approach, the new approach saves one β unit. The memory requirement and overall latency are exactly the same. Furthermore, the forward recursion state metrics are exactly the same for both cases. However, it is easy to see that, on the average, the backward recursion state metrics computed using the type-II scheme are more reliable than those computed using the SSW approach as the average recursion depth (of backward recursion computation) of the new approach is larger. It should be noted that the dashed arrows outside the decoded frame represent the case when the current decoding frame is a sub-frame of a large frame. In that case, a hybrid parallel decoding scheme is employed. The details will be given later.

Although the type-II parallel decoding scheme is superior to the type-I scheme in the case of 2-level parallelism, it is not suitable for higher levels ($F > 2$) of parallelism due to the following problems: (1) the total number of computation units is not minimized, (2) the requirement for storage of state metrics is not minimized, and (3) the overall decoding cycles are not minimized.

Shown in Fig. 5 is the timing diagram for a 3-parallel type-I parallel decoding scheme. For simplicity and clarity, only 21 sliding blocks are drawn in the diagram. All symbols in the diagram have the same meanings as in Fig. 3. The left-hand side of the thick dash-dot line is the part which can be extended. For example, if the total number of sliding blocks increases by $3 * W$ where W represents an integer, then the left-hand side of the dash-dot line will be extended by $3 * W$ sliding blocks. The right-hand side of dash-dot line is the removable part, *i.e.*, the last one or two columns can be removed if the overall sliding blocks is not multiple of 3. It should be noted that, in this decoding scheme, the right-hand side of the dash-dot line can be moved into the beginning of the frame. In that case, this decoding scheme has a small benefit as the type-I 2-level parallel decoding scheme discussed above.

It can be seen that totally 4 α computation units, 4 β units and 3 LLR/L_{ex} computation units are used. The state metrics are required to store as large as three sliding blocks. In general cases, the required overall decoding cycles are $S + 2$. Compared with the SSW approach, the new approach requires 1 more α unit, but saves 2 more β units. As there is no difference in real implementation between α and β units, the net saving for the new approach is 1 computation unit (α or β). The latency and memory requirement for both approaches are exactly the same.

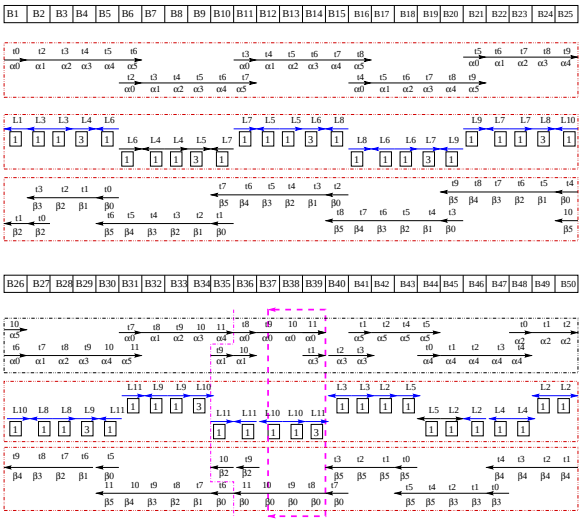

Figure 6. A type-I 5-parallel turbo decoding scheme

For $F > 3$ cases, the type-I parallel decoding scheme also runs into problems. Shown in Fig. 6 is an example of a 5-parallel decoding scheme. Totally 6 α units , 6β units and 4 LLR/L_{ex} computation units are used. The overall decoding cycles are $S + 2 = 12$.

Compared with the SSW approach, this scheme has a net saving of 3 α units. The latency is the same. However, the storage of state metrics is required for 7 sliding blocks. That's to say the new scheme requires storage of state metrics of additional 2 sliding blocks. There are two ways to trade off the saving of computation units for the storage requirement, interested readers are referred to [7].

2.3. Hybrid parallel decoding schemes

If a very high level parallelism is required, hybrid parallel decoding schemes have to be employed as neither type-I nor type-II parallel decoding scheme is suitable.

If a 4-parallel decoding scheme is required, we can divide a frame into 2 parts as we did with the SSW approach and then apply the type-II 2-parallel decoding scheme onto each part. In this case, two computation units can be saved compared with the SSW approach.

If a 5-parallel decoding scheme is required, we can either use the type-I approach combined with a trade-off method as in [7], or use a hybrid parallel decoding scheme. In the later case, a frame is divided into two parts, *i.e.*, part A and part B, with a reasonable overlap depth. Then the type-II 2-parallel decoding scheme is used for Part A and the type-I 3-parallel decoding scheme is employed for Part B. In this case, two computation units can be saved. The overall saving is about the same as the type-I 5-parallel decoding scheme with a good trade-off.

For a 6-level parallelism decoding, a frame is divided into three equivalent sub-frames. The type-II 2-parallel decoding scheme is applied onto each sub-frame. The overall net saving is three computation units, which is obviously better than applying the type-I 3-parallel decoding scheme onto two equivalent sub-frames.

For $F \geq 7$ cases, hybrid decoding schemes can be constructed similarly as we did in the above. For the optimal choice, the type-I decoding scheme should be applied as many as possible.

3. APPLICATION OF PIPELINE-INTERLEAVING

Pipeline interleaving technique was proposed for high-speed recursive filter design in [8]. In turbo decoders, only α and β computation units have recursive loops. Using pipeline interleaving, one hardware unit can perform operations previously assigned to two or more (depending on pipelining stages) hardware units. In general cases, the hardware overhead introduced by pipeline registers and interleaving MUX's are much smaller than the saved hardware units. So, the total area would be reduced in general.

Pipeline registers can be placed inside the loop of α/β computation units. In order to maintain the same throughput, the clock cycle for the recursive loop has to be increased by ν times, where ν denotes the number of stages of pipeline interleaving. As the dynamic power dissipation is proportional to the clock frequency and the total charge capacitance in the circuit, the pipeline-interleaving architecture will consume slightly more power than the original architecture in which multiple identical hardware units are used due to its small overhead [9]. However, as the area overhead (related to the total number of transistors) is reduced, the total leakage power and short circuit power could be reduced.

4. PERFORMANCE COMPARISON

Simulation for the $1K3$ case was performed assuming an AWGN channel with BPSK modulation and code rate r=1/2. (due to lack of space, the simulation results of $1K5$ are not presented here). $8M$ information bits have been generated and the results shown in Fig.7 are performance comparison of three different parallel decoding schemes under $E_b/N_o \leq 3.0dB$. The dashed line represents the type-I 2-parallel decoding scheme. The dash-dot line represents the case using global sliding window approach. The solid lines represents the case using type-I 4-parallel decoding scheme. It can be seen that the performance of 2-parallel decoding scheme is a little worse than the case with non-segmented sliding window approach while the performance of 4-parallel decoding scheme is a little better. It can be deduced that the performance of pipelined parallel decoding schemes are at least as good as SSW approach when the parallelization is no less than 3.

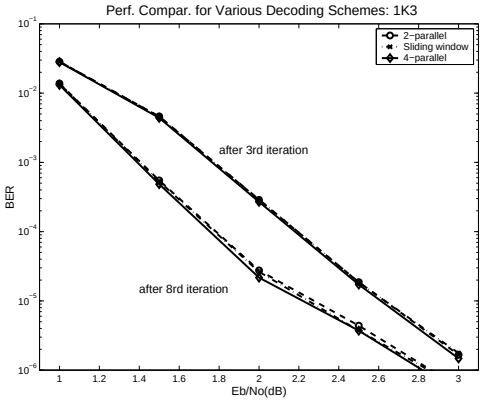

Figure 7. Performance comparison of various decoding schemes: 1K3 case

5. CONCLUSIONS

In this paper, two types of area-efficient parallel decoding schemes are proposed to increase the decoding throughput of MAP turbo decoders. Detailed comparison about storage requirement, number of computation units and the overall decoding latency has been provided for various parallel decoding schemes. Hybrid parallel decoding schemes have been shown to be the best choices for very high level parallelism cases. Simulation results have shown that the proposed area-efficient parallel decoding schemes have no performance degradations in general. The pipeline-interleaving technique can be combined with area-efficient parallel decoding schemes to achieve good trade-offs between area and power.

6. REFERENCES

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

[2] J. Hsu and C. Wang, "A parallel decoding scheme for Turbo codes", *IEEE Symp. on Circuits and Systems, Geneva, Switzerland*, pp. 445–8, 1998.

[3] A. Worm, H. Lamm, and N. Wehn, "A high-speed map architecture with optimized memeory size and power consumption", *in IEEE Workshop on SiPS*, pp. 265–274, 2000.

[4] A.J. Viterbi, "An intuitive justification of the MAP decoder for convolutional codes", *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 260–264, February 1998.

[5] Z. Wang, H. Suzuki, and K. K. Parhi, "VLSI implementation issues of Turbo decoder design for wireless applications", *IEEE Workshop on Signal Processing Systems, Design and Implementation*, pp. 503–512, Oct. 1999.

[6] H. Suzuki, Z. Wang, and K. Parhi, "A K=3, 2Mbps Low Power Turbo Decoder for 3rd Generation W-CDMA Systems", *Proceedings of the IEEE 1999 Custom Integrated Circuits Conference*, pp. 39–42, 2000.

[7] Z. Wang, "High Performance, Low Complexity VLSI Design of Turbo decoders", Ph.D. Dissertation, U of Minnesota, Twin Cities, Sept. 2000.

[8] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters-part I: pipelining using scattered look-ahead and decomposition", *IEEE Trans. on ASSP*, vol. 37, July 1989.

[9] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation, Wiley, NY*, 1999.